

ISEP
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

LICENCIATURA
RAMO DE COMPUTADORES E SISTEMAS
PROJECTO

A PLATAFORMA .NET

Autora
Natália Sofia Vicente Correia
Nº 930507

Orientador
Engº Paulo Ferreira

Data de realização
Setembro 2001

Índice

1	Introdução.....	1
2	.Net.....	3
2.1	O que é o .Net.....	3
2.2	A solução .Net.....	3
2.3	Como irá o .Net afectar a maneira como faço o meu trabalho?.....	4
2.3.1	Utilizadores.....	4
2.3.2	Administradores de rede.....	4
2.3.3	Programadores.....	4
2.4	A estratégia do .Net.....	5
3	A plataforma .Net.....	7
3.1	Critérios de análise.....	8
4	O Framework .Net.....	11
4.1	Serviços web.....	11
4.1.1	A arquitectura do serviço web.....	13
4.2	XML.....	14
4.3	SOAP.....	14
4.4	Os objectivos do Framework .Net.....	15
4.5	As três partes do Framework .Net.....	16
4.5.1	A common language runtime.....	16
4.5.1.1	As principais características.....	17
4.5.2	As classes unificadas (o que programador vê).....	19
4.5.3	ASP .Net – Aplicações web.....	20
4.5.3.1	Dentro da ASP .Net.....	21
4.6	Produtividade do programador.....	22
5	Visual Studio .Net.....	27
5.1	Os objectivos.....	27
5.2	As características.....	28
5.3	As ferramentas da produtividade.....	28
5.4	Construir a web programável.....	37
5.4.1	Forms web.....	37
5.4.1.1	Criação de uma form web.....	37
5.4.2	Serviços web.....	38
5.4.2.1	Criar.....	38
5.4.2.2	Invocar.....	40
5.4.2.3	Consumir.....	41
5.5	Visual Basic .Net.....	42
5.5.1	Inovações na linguagem.....	42
5.5.1.1	Herança.....	42
5.5.1.2	Overloading.....	43
5.5.1.3	Construtores parametrizáveis.....	43
5.5.1.4	Tratamento de excepções estruturadas.....	43
5.5.2	Características adicionais da linguagem modernizada.....	44
5.5.2.1	Verificação severa de tipos.....	44
5.5.2.2	Membros distribuídos.....	45
5.5.2.3	Inicializadores.....	45
5.5.3	A ferramenta de upgrade.....	45

5.5.4	Upgrade de conhecimentos.....	47
5.6	Visual C++ .Net.....	50
5.6.1	Poder e flexibilidade.....	50
5.6.2	Servidor ATL.....	50
5.6.2.1	Como funciona o servidor ATL.....	51
5.6.2.2	Performance e escalonabilidade.....	52
5.6.2.3	Um melhor suporte do servidor de aplicações.....	52
5.6.2.4	Totalmente configurável.....	52
5.6.3	Extensões manejáveis para o C++.....	53
5.6.3.1	O que são extensões manejáveis?.....	53
5.6.3.2	Uma migração suave do código existente para o Framework .Net.....	54
5.6.3.3	Aceder a um componente C++ de uma linguagem Framework .Net.....	54
5.6.3.4	Aceder a classes Framework .Net a partir do código nativo.....	54
5.6.4	Programação de atributos.....	54
5.6.4.1	Tipos de atributos.....	55
5.6.4.2	Utilizar os atributos.....	55
5.6.5	Escrever código robusto.....	56
5.6.6	Performance pura.....	56
5.7	C#.....	57
5.7.1	Hello World.....	57
5.7.2	Tipos.....	59
5.7.2.1	Tipos predefinidos.....	59
5.7.2.2	Tipos de arrays.....	59
5.7.2.3	Tipo system unification.....	59
5.7.3	Classes.....	59
5.7.3.1	Campos.....	60
5.7.3.2	Métodos.....	60
5.7.3.3	Propriedades.....	60
5.7.3.4	Eventos.....	61
5.7.3.5	Indexadores.....	61
5.7.3.6	Destruidores.....	61
5.7.3.7	Herança.....	61
5.7.3.8	Structs.....	61
5.7.3.9	Interfaces.....	61
5.7.3.10	Delegados.....	61
5.7.3.11	Enums.....	62
5.7.3.12	Namespaces e assemblies.....	62
5.7.4	Atributos.....	62
5.8	Forms Windows.....	62
5.8.1	Criar uma aplicação forms Windows.....	63
5.8.2	Herança visual.....	63
5.8.3	Desenho de form preciso.....	64
5.8.4	Um custo de posse mais baixo.....	64
5.9	Quais são os requisitos do sistema para o Visual Studio .Net Beta 1?	65
6	Segurança.....	67
6.1	Segurança baseada em funções.....	67
6.2	Segurança de aplicações web.....	67
6.3	Segurança baseada em factos.....	68

6.4	Criptografia.....	68
7	Conclusão.....	71

1 Introdução

Quando foi criada, a *web* era basicamente um sistema de ficheiros para leitura com o benefício de que utilizou standards e protocolos da industria, possibilitando acesso fácil ao conteúdo dos ficheiros. Os poucos *sites* da *web* que eram interactivos eram tipicamente extensões externas de aplicações de duas camadas existentes.

Os primeiros desenvolvimentos para a *web* eram desenvolvidos tipicamente com a linguagem de programação *C* e com *CGI* (*Common Gateway Interface*), com os quais muitos programadores tinham pouca experiência. Resultado, os custos de desenvolvimento para aplicações *web* dinâmicas eram altos.

Mais, a maioria destas aplicações *web* eram construídas com arquitecturas de duas camadas, causando desafios para a escalonabilidade e integração da aplicação. Os programadores simplesmente não desenhavam aplicações *web* para serem utilizadas unicamente pelas páginas *web* ai guardadas ; ou seja, a interface do utilizador e a lógica da aplicação eram a mesma coisa. Consequentemente, era difícil ligar aplicações *web* de maneira a formarem conjuntos mais interessantes. Um exemplo deste problema, seria um *site* que vendia varões de cortinas, mas não oferecia cortinas, forçando assim potenciais clientes a visitar pelo menos dois *sites* diferentes para comprarem uma solução completa para as suas janelas.

Como resultado dos avanços no *COM* (*Component Object Model*) da *Microsoft* e o lançamento de tecnologias tais como as páginas *Active Server (ASP)* da *Microsoft* em 1996, os *sites* da *web* ofereciam uma experiência mais interactiva ao utilizador. As *ASP* fazem isso, tornando mais fácil (através linguagens *script* simples), a ligação entre a lógica do comercio e os serviços da plataforma, de que os programadores necessitam. O suporte *COM* torna mais fácil escrever aplicações através da sua habilidade de “*empacotar*” esta lógica de negócio em unidades modulares que podem ser escritas numa grande variedade de populares linguagens de programação, tais como o *Visual Basic* da *Microsoft*, o *C++* ou o *COBOL*.

Os *sites* da *web* estão agora a oferecer experiências ao utilizador mais ricas, e estão a tomar os passos mais básicos para superar algumas das dificuldades da integração das aplicações, com truques tais como utilizar *frames HTML* para embutir o *site* de uma empresa, noutra.

Até a data, as empresas que tentaram oferecer soluções para permitir que um *site* na *web* pode-se expor uma aplicação com informação e funcionalidades integradas de uma forma modular, escalonável e “Internet amigável”, têm encontrado alguns desafios. Alguns dos desafios mais importantes são:

- tempo de comercialização. O tempo de desenvolvimento necessário para se colocar uma aplicação ou um *site* (da *web*) no mercado pode ser demais, para uma procura que deixou (entretanto) de existir.
- graduação da *web*. Os modelos de objectos e os componentes existentes, simplesmente não funcionam com os protocolos da Internet. O desenvolvimento de aplicações solitárias, que podem ser encaminhadas e utilizadas por qualquer servidor, é um conceito estranho a muitos programadores. No entanto um modelo deste género é necessário para se obter escalonabilidade global.

- falta de ferramentas de desenvolvimento. As ferramentas disponíveis presentemente, não oferecem às organizações a flexibilidade necessária para se poderem manter à frente dos seus competidores. Num mundo de rápidas mudanças como é o da Internet, parceiros podem tornar-se competidores diariamente e as ferramentas de desenvolvimento necessitam de oferecer a agilidade para se resolverem estes problemas.

E é assim que aparece a plataforma *.Net*.

Neste trabalho, pretende-se dar uma apresentação desta plataforma de que se fala nos dias de hoje, fazendo um estudo, não exausto das suas partes mais importantes e interessantes.

2 .Net

2.1 O que é o .Net?

O .Net é melhor descrita como uma iniciativa que vai permitir á Internet, ser a base de um novo sistema operativo.

Irá-nos libertar dos constrangimentos do hardware, facultando a informação para o utilizador na Internet.

O .Net torna-se assim importante para os utilizadores, porque torna toda a sua informação disponível, através de todos os dispositivos.

Também é importante para os programadores, porque irá mudar a maneira como desenvolvem aplicações, permitindo-lhes ligarem-se a serviços *web* (designados em inglês por *web services*).

De uma maneira simples, o .Net é a nova estratégia da Microsoft para distribuir software como um serviço.

2.2 A solução .Net

O .Net, é a plataforma da Microsoft para serviços *web* em *XML*. A .Net contém tudo o que é necessário para construir e “correr” *software* baseado em *XML*, a *lingua franca* da troca de dados na Internet.

O .Net da Microsoft resolve vários problemas que afectam o desenvolvimento de *software* hoje em dia:

- Interoperabilidade, integração e extensibilidade das aplicações, são demasiado difíceis e dispendiosos. A confiança do .Net no *XML* – um standard gerido pelo *World Wide Web Consortium (W3C)* – remove barreiras em relação á distribuição de informação e integração de *software*.
- Os desafios de integração, são compostos pelas inúmeras tecnologias de *software* concorrentes que infestam a industria. O .Net, é feito com standards “abertos” e abrange todas as linguagens de programação.
- O que os utilizadores finais sentem quando utilizam o seu *software*, não é simples ou suficientemente apelativas. Os utilizadores finais sentem-se frustados porque estão impossibilitados de disponibilizarem dados com facilidade entre as suas aplicações, ou actuarem de acordo com informação quando a conseguem aceder. O *XML* torna a troca de dados fácil, e o *software* .Net dá-lhes a possibilidade de trabalhar com os dados assim que são recebidos.
- Os utilizadores finais não controlam a sua informação e dados pessoais quando trabalham na *web*, o que os leva a ter preocupações em relação á privacidade e segurança. O .Net providencia uma serie de serviços, que permite aos utilizadores manusearem a sua informação pessoal e controlarem o acesso a essa mesma informação.
- Os programadores de “*COMs*” e de *sites web* têm dificuldade em providenciar informação suficiente aos utilizadores, principalmente devido ao facto de as

suas aplicações e serviços não serem compatíveis a outras e actuarem como “ilhas” de informação desligadas umas das outras. O *.Net* foi desenhado para possibilitar a agregação de valores e serviços de múltiplos *sites* e empresas em experiências coerentes para os utilizadores.

Os sistemas operativos *MS-DOS* e *Windows* mudaram significamente a computação, e assim também será com o *.Net*. O *MS-DOS* levou à aceitação dos computadores pessoais para os negócios e para as casas; o *Windows* elevou a interface gráfica para o utilizador, como a maneira preferida de actuar com software, e a interface gráfica tornou a computação pessoal (personal computing) mais forte.

2.3 Como irá o *.Net* afectar a maneira como faço o meu trabalho?

De momento, não irá afectar nada. Para já o *.Net* é apenas uma visão, não uma realidade. Contudo, se tudo correr como planeado, o *.Net* tem potencial para afectar três grupos de pessoas: os utilizadores, os administradores de redes, e os programadores. Sendo assim, trabalhar no café da esquina será como se estivesse a trabalhar em casa.

2.3.1 Utilizadores

Esta plataforma flexível, significa que os dados do utilizador serão automaticamente adaptados ao seu ambiente de trabalho: os mesmos dados terão um aspecto diferente num desktop cliente e num pequeno browser de um PC portátil.

2.3.2 Administradores de rede

O que é que este modelo tem para oferecer a um administrador de rede? As aplicações *.Net* são supostas serem carregadas para as suas próprias directórias, independentes umas das outras. Isto irá ter dois efeitos. Primeiro, seguindo o modelo “dividir dados, não código” que a *Microsoft* tem estado a endossar, iria significar o fim do “inferno das *DLLs*”, onde diferentes aplicações sobreescrivem as bibliotecas auxiliares com o mesmo nome. Em segundo, as aplicações deveriam em princípio não tocar no *Registry*, resolvendo assim o problema de “tatuagem” no *Registry* que lá deixam *settings* no *Registry* mesmo depois de se desinstalar as aplicações. Mais, as aplicações *.Net*, irão usar segurança baseada no certificado *Kerberos*, para assegurar que só pessoal autorizado pode aceder aos dados.

2.3.3 Programadores

Para os programadores, o *.Net* pode tornar possível criar aplicações que obtêm informação de fontes sem qualquer tipo de relação, utilizando qualquer que seja a linguagem, ou seja utilizam aquela com que se sintam mais à vontade. Os programadores não vão ter de alterar os modelos de programação de objectos *COM* para *DLLs*, não vão ter de lutar com uma variedade de ferramentas para *debug* ou escrever pacotes de instalação para as aplicações *.Net*. Noutras palavras, os programadores deverão poder escrever as suas aplicações na linguagem que

desejarem, criar os ficheiros de suporte que a aplicação necessita, e **esperar** que ela funcione.

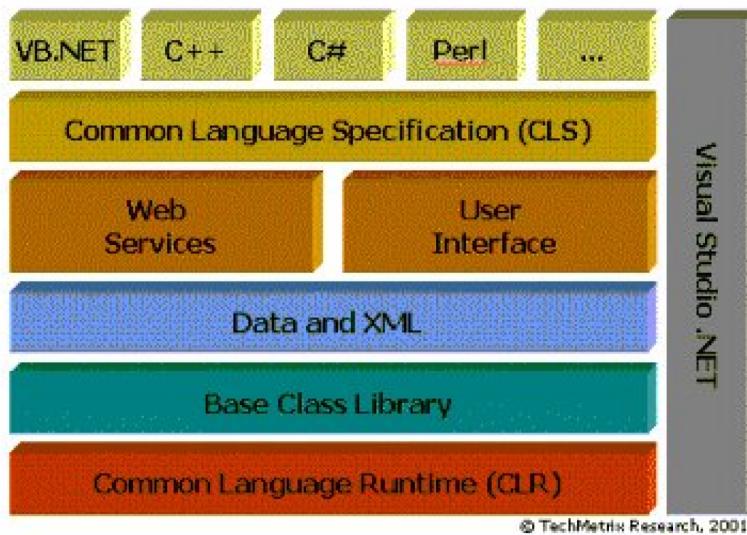


Figura 2.1 – A arquitectura geral .Net

O .Net não vai acontecer imediatamente. Os browsers necessitam de analisadores *XML* de confiança, os servidores necessitam de máquinas que transformam informação “bruta” em *XML*. Dados que não estão formatados em *XML* não estarão disponíveis para as aplicações.

2.4 A estratégia do .Net

Em vez de focarem a suas atenções em aplicações isoladas para servidores e clientes, os programadores .Net escrevem serviços *web XML*. Depois juntam esses serviços em “constelações” de *software*, “fracamente ligadas” (*loosely coupled*), utilizando mensagens *XML* para comunicar entre os serviços *web XML*. Para conseguirem isto, os programadores precisam de:

- uma plataforma de *software* para construir uma nova experiência pessoal e integrada, para os utilizadores;
- um modelo de programação, e ferramentas para construir e integrar serviços *web XML*;
- um conjunto de serviços programáveis para proporcionar a fundação para aplicações e serviços.

A estratégia do .Net abrange estes três pontos.

O .Net inclui:

- **A plataforma .Net**, que é um conjunto de ferramentas de programação e uma infra-estrutura que permite a criação, distribuição, administração e agregação de serviços *web XML*.

- **As experiências *.Net*,** que são os meios com que os utilizadores finais interagem com o *.Net*.

3 A Plataforma .Net

A plataforma .Net contém as ferramentas que são necessárias para criar e correr serviços web *XML*. Tem quatro componentes:

1. O *Framework .Net* e o *Visual Studio .Net*.

Estas são as ferramentas de desenvolvimento para construir serviços web *XML*. O *Framework .Net*, é um conjunto de interfaces de programação, no coração da plataforma .Net; o *Visual Studio .Net* é um conjunto de ferramentas de programação, multi-linguagens.

2. Infra-estrutura do servidor.

A infra-estrutura do servidor .Net, incluindo o *Windows*, é um conjunto de aplicações de infra-estrutura para construir, distribuir e operar serviços web *XML*. Algumas tecnologias chave incluem suporte para *XML* e a orquestração do processo negocial, tanto para aplicações, como para serviços. Estes servidores incluem:

- *Application Center 2000*, para permitir soluções *scale-out*;
- *BizTalk Server 2000*, para criar e administrar a orquestração do processo negocial, tanto para aplicações como para serviços;
- *Host Integration Server 2000*, para aceder a informação e aplicações no *mainframe*;
- *Mobile Information 2001 Server*, para permitir a utilização das aplicações nos dispositivos móveis tais como telemóveis;
- *SQL Server 2000*, para guardar e reaver dados *XML* estruturados.

3. Os serviços de construção em bloco.

Os serviços de construção em bloco são um conjunto de serviços web *XML*, centrados no utilizador, que fazem a transição do controlo da informação, entre a aplicação e os utilizadores, revirando a *web* de dentro para fora, e possibilitando, simplicidade e consistência personalizadas, através das aplicações, serviços e dispositivos enquanto asseguram que o consentimento do utilizador é a base de todas as transações. Eles incluem o *passaport* (para a identificação do utilizador) e serviços para a entrega de mensagens, arquivamento de ficheiros, administração das preferências do utilizador, manuseamento do calendário, e outras funções. A *Microsoft* vai oferecer alguns serviços de construção em bloco, em áreas críticas da infra-estrutura do .Net; um grande leque de parceiros e programadores irão aumentar significativamente o conjunto dos serviços de construção em bloco.

4. Os dispositivos inteligentes.

O .Net utiliza *software* para os dispositivos inteligentes para permitir que *PCs*, portáteis, *workstations*, telefones inteligentes, consolas de jogos e outros dispositivos inteligentes, possam operar no universo .Net.

Um dispositivo inteligente é:

- inteligente sobre o utilizador: utiliza a identidade, *profile*, e dados .Net, do utilizador, e simplifica a experiência do utilizador; é inteligentes sobre a presença do utilizador e permite a construção de notificações, em resposta a presenças ou faltas desta.

- inteligente sobre a rede: da resposta a constrangimentos de largura de banda; proporciona suporte para utilização de aplicações *online* e *offline*; comprehende que serviços estão disponíveis.
- inteligente sobre informação: aceder, analisar e actuar sobre dados, em qualquer lado, a qualquer altura.
- inteligente sobre outros dispositivos: descobre e notifica PCs, dispositivo inteligentes, servidores e a Internet; sabe proporcionar serviços a outros dispositivos.

Tendo agora dado uma breve explicação dos quatro componentes de que consiste a plataforma .Net, irei fazer um estudo mais profundo do primeiro componente da plataforma: o *Framework .Net* e o *Visual Studio .Net*.

3.1 Critérios de análise

Se examinar-mos as necessidades dos programadores e das organizações, podemos criar uma lista de critérios para avaliar o *Framework .Net* e o *Visual Studio .Net*. Alguns desses critérios são:

- Produtividade do programador, durante toda a extensão dos desafios de desenvolvimento, no cliente, no servidor e na integração de tudo o que se encontra no meio (do cliente e do servidor). As ferramentas de desenvolvimento rápido de aplicações, já são á muito, aceites como um standard na industria de desenvolvimento. Para se obter sucesso nesta área, a ferramenta de desenvolvimento deverá proporcionar ferramentas rápidas através da plataforma de aplicação distribuída na sua totalidade, ou seja, clientes, servidores e o desdobramento do ambiente.
- Um modelo completo para integração de aplicações, integrado na ferramenta de desenvolvimento e não um *patch* dispendioso. Hoje em dia os clientes não estão á procura de um único ambiente para substituir os seus sistemas existentes. Eles necessitam de uma plataforma que está preparada para ser facilmente ligada nos ambientes existentes e variados *landscapes IT*.
- Suporte para as competências existentes dos programadores. Os clientes descobriram os custos exorbitantes de achar programadores competentes em tecnologias específicas. Analistas de industria proeminentes, recomendam utilizar as competências existentes, e a evitar os custos altos e a alta percentagem de insucesso da dependência das competências técnicas em substitutos temporários.
- Suporte para criação e integração de componentes criados em diversas linguagens de programação. No mundo real, as empresas têm investimentos numa variedade de linguagens de programação e competências de pessoal. Sem qualquer tipo de suporte que abrange este investimento existentes as empresas são forçadas a substituir ou re-treinar o seu pessoal, uma perspectiva cara e pouco amigável.
- Aderência a standards de Internet. O meio de comunicação mais bem sucedido na historia da humanidade, tem sido a Internet, e foi assim, só por causa dos processos colaborativos que surgiram, em quais os diferentes partidos concordaram em como comunicariam. Na época de componentes de *software* da *web* programáveis e inter-operáveis, ter standards é a única maneira para assegurar que todos os jogadores podem participar.

- Um modelo de programação consistente e unificado, para clientes baseados na web, abrangentes, bem como interfaces ricas e dispositivos inteligentes emergentes. Hoje em dia, os clientes estão a avaliar toda a gama de dispositivos (para clientes) existentes, incluindo *PC*, *browser*, *PDA*, e telemóveis. Só tendo um modelo de programação que suporte, elegantemente, todas as experiências dos utilizadores, e aquelas que emergirão no futuro, e que contenha um modelo simples para manter uma lógica de negócio constante, através de qualquer arquitectura de cliente. Só então os complexos sistemas distribuídos serão bem sucedidos.
- Uma arquitectura aberta, que permita ter capacidades “enterprise”, bem como as novas agilidades “enterprise”. As ferramentas de desenvolvimento precisam de desenvolver aplicações que são de confiança, inter-operáveis, manejáveis, e seguras. Ao mesmo tempo as ferramentas devem conceder às organizações a flexibilidade necessária para chegar ao mercado o mais rapidamente possível e manterem-se à frente dos seus competidores.

O *Framework .Net* e o *Visual Studio .Net*, satisfazem estes critérios completamente, e mais eficientemente que qualquer outro produto disponível neste momento. São precisamente estes critérios, que farão os clientes serem pioneiros nas suas respectivas indústrias, para construir e manter aplicações melhores, e trazer novo *software* e serviços para o mercado mais rápido e menos dispendioso que os seus competidores.

4 O Framework .Net

O *Framework .Net* é a infra-estrutura para a plataforma .Net. A *common language runtime* e as livrarias de classes, combinam para providenciar serviços e soluções que podem ser facilmente integradas dentro e através de uma variedade de sistemas.

O *Framework .Net* é o resultado de dois projectos. O objectivo do primeiro projecto era melhorar o desenvolvimento do sistema operativo *Windows*, tentando especificamente melhorar o *COM* (o *Microsoft Component Object Model*). O segundo projecto apontava para a criação de uma plataforma para entrega de software como um serviço. Estes dois projectos juntaram-se á três anos atrás. O produto final melhora drasticamente a produtividade do programador, é de fácil integração, é uma aplicação de confiança e introduz um conceito de computação completamente novo: serviços web (*web services*) – aplicações “fracamente ligadas” e componentes desenhados para o ambiente de computação heterogéneo existente, que comunica utilizando protocolos de Internet standards, tais como *XML* e *SOAP*.

Antes de prosseguir com o estudo do *Framework .Net*, acho importante falar de três termos que já foram mencionados e que irão certamente, continuar a ser utilizados com muita frequência quando se falar do *Framework .Net*. Eles são:

- Serviços *web*;
- *XML* e
- *SOAP*.

4.1 Serviços *web*

Um serviço *web* pode ser pensado como uma chamada a um procedimento remoto (*remote procedure call*), ordenada em *XML* sobre o *HTTP*. A promessa de um serviço *web*, é para que todas as aplicações possam ser expostas como um serviço na Internet. Os serviços *web* são uma chave para a tecnologia, porque são baseados em mecanismos que já foram provados e que são utilizados no dia-a-dia.

O transporte *HTTP*, já é um caminho bem estabelecido, mesmo através de *firewalls* corporativas. O *SSL*, é uma utilização muito espalhada para a segurança *HTTP*. Qualquer linguagem, modelo de componente e sistema operativo pode utilizar e disponibilizar serviços *web*.

Os serviços *web* são módulos de software, que são construídos utilizando o *XML* para a troca de dados, para ajudar as aplicações, os serviços, e os dispositivos a trabalharem em conjunto. Dividir dados através do *XML*, permite que eles se tornem independentes uns dos outros e simultaneamente dando-lhes a possibilidade de se interligarem, “levemente”, num grupo que realiza uma tarefa em particular.

A maneira mais fácil de perceber como funcionam os serviços *web*, é comparando-os a peças de *Lego*. Os serviços *web* são unidades independentes tais como as peças de *Lego*. As peças de *Lego* têm uma maneira standard de se “juntarem”, tal como os serviços *web* – através de mensagens *XML*. Quando juntamos as peças de *Lego*, construi-se um objecto: uma casa, um barco, uma girafa ou um avião. Quando juntamos serviços *web*, construi-se uma solução de software que realiza uma

determinada tarefa. Tal como podemos utilizar a mesma peça de *Lego*, para ser parte de diferentes objectos, podemos utilizar um único serviço *web* em muitos grupos diferentes, como parte de uma solução para muitas tarefas diferentes.

“Um serviço web é tal e qual como uma aplicação”. Isto torna claro, que os serviços *web* não são nada mais que aplicações. Sendo assim, proporciona funcionalidades que as aplicações habituais proporcionam.

“São entregues como um serviço”. Pode-se comparar os serviços *web* aos serviços do *Windows NT*, por isso os serviços *web* também contém as funcionalidades de serviços. Correm em *background* e esperam que os clientes se “liguem” a eles. Como é um serviço, liberta o programador de escrever código para o serviço *web* “ouvir” múltiplos clientes - fá-lo automaticamente.

Os serviços *web* também tornam possível aos programadores, escolher entre construir e comprar os “bocados” das suas aplicações, ou entre consumir outros serviços *web* para completar as suas soluções ou expor os seus próprios serviços para outras aplicações ou serviços. Isto significa que uma empresa não tem que fornecer todas as peças de uma solução para o cliente.

Para além de os serviços *web* serem independentes uns dos outros, também são independentes do dispositivo utilizados para acede-los. Ao contrário das aplicações *standalone*, os serviços *web* não estão amarrados a uma língua de programação particular, a uma única aplicação, ou serviço *online*. Isto permite aos utilizadores finais, a liberdade de poder escolher o dispositivo em que preferem trabalhar, desde um computador pessoal aos dispositivos inteligentes, tais como telemóveis. Podem ser invocados por um *browser* qualquer.

Comunicação através de mensagens. A interface dos serviços *web* pode ser descrita estritamente em termos de mensagens. Um serviço *web* aceita e gera. Isto significa que a comunicação com serviços *web*, é feita através de mensagens, sendo o *XML* o formato escolhido para as mensagens.

Uma vez que os serviços *web* estejam em uso, poderá ocorrer o problema de descobrir o serviço *web* certo de que necessitamos. Para resolver este problema os serviços *web* providenciam ficheiros *Service Discovery (DISCO)* (uma outra especificação disponível com os serviços *web* é o *Service Discovery (DISCO)*). *Service Discovery* é um caminho standard a ser utilizados pelos fornecedores de serviços *web*, para disponibilizar contractos de serviços. Isto permite aos programadores descobrirem facilmente os contractos de serviços para os serviços *web*.

4.1.1 A arquitectura do serviço web

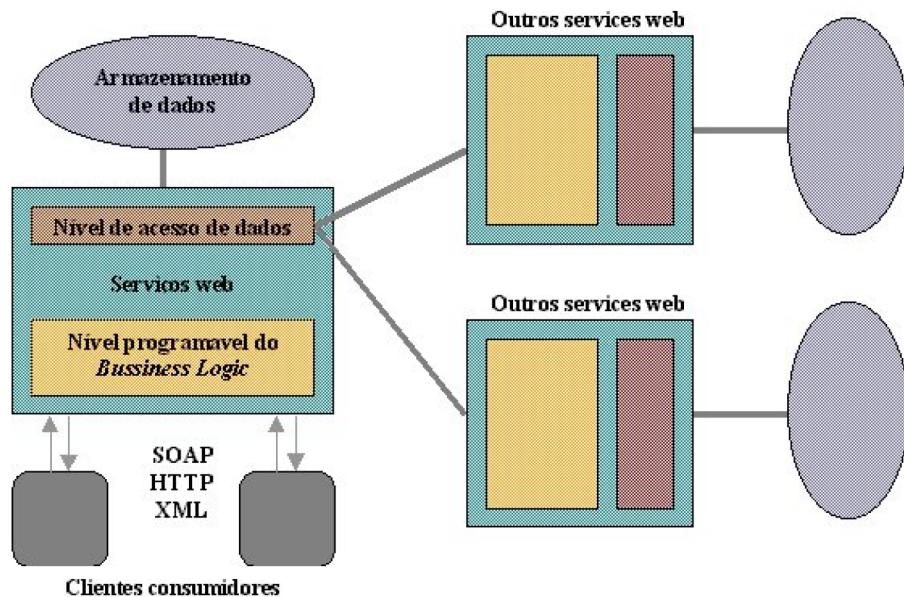


Figura 4.1 – A arquitectura do serviço web

Os serviços web são aplicações web programáveis, que contêm *bussiness logic* (“lógica do negócio”) acessível através de protocolos de Internet standard. A arquitectura do serviço web é baseada na arquitectura do *Windows DNA 2000*. É uma arquitectura de vários níveis. Na base está a persistente base de dados. A base de dados não está em contacto com o cliente em qualquer altura, mantendo assim a base segura de qualquer maldade. O nível de acesso aos dados, actua como ligação entre a base de dados persistente e o serviço web.

O nível de acesso aos dados pode também utilizar dados de outros serviços web.

Por cima do nível de acesso aos dados está o tão importante nível do *bussiness logic* (“lógica do negócio”). Pode-se dividir este nível em múltiplas partes, de acordo com as necessidades de cada um. Por exemplo, um dos níveis podia ser responsável pela validação da base de dados, e um outro nível poderia formatar os dados de acordo com as necessidades de cada um. Este nível (*bussiness logic*) contém toda a lógica que deverá ser executada para processar os dados “crus” do cliente, ou do nível de acesso aos dados.

Este nível, acima descrito, não contém componentes *UI*. Estes são componentes programáveis, que deverão ser optimizados para aguentar com a carga de pedidos. Eles também devem ser tolerantes a falhas e devem conseguir recuperar de falhas rapidamente. Como a maior parte do trabalho é feito no nível do *bussiness logic* (“lógica do negócio”), devem ser escalonáveis e rápidos.

Por cima destes níveis, está o nível *listener* (“ouvinte”). Este nível está à escuta de pedidos de clientes para dados e contractos. É este nível, que é responsável por descodificar os dados entregues pelos clientes e empacotar os dados do nível *bussiness logic* (“lógica do negócio”) e transmiti-los ao cliente. Este nível, é o único nível no serviço web que conhece a sua parte no serviço web. Quando um serviço web é requisitado, o nível *listener* (“ouvinte”) é o único nível que fica em contacto com o

cliente. Este nível foi construído dentro do serviço *web*, por isso não existe nenhum tipo de preocupação em implementá-lo.

4.2 XML

XML, ou a *eXtensible Markup Language*, é tão fundamental para o *.Net* como a língua que falamos e escrevemos, é para a nossa comunicação.

O *XML* é a *lingua franca* do *.Net* e é a base para tudo o que o *.Net* é, e virá a ser. As bases de dados irão ler e escrever *recordsets* em *XML*. *Browsers* da *web* irão aceitar o *XML* e exibi-lo, quando estiver acompanhado de *style sheets*. O *Visual Studio* irá gerar código *XML*.

Hoje em dia, o *XML* é tão ubíquo como os servidores *web*. Quase todas as plataformas de computação, têm a capacidade de processar *XML* e podem assim retirar o conteúdo de um documento *XML*. O *Windows* pode. O *Linux* pode. *MVS* e *VMS* também podem. Até os telemóveis podem. Por isso, se for possível arranjar uma maneira de enviar um documento *XML* para um sistema remoto, é bem possível que se consiga fazer sentido dos dados que o documento contém. É por isso que os documentos *XML* são enviados utilizando *SOAP*.

4.3 SOAP

SOAP/XML é verdadeiramente o sangue das veias dos serviços *web*, utilizando uma linguagem (*XML*) e protocolo (*SOAP*) universal para descrever o significado dos dados.

SOAP (*Simple Object Access Protocol*), é basicamente um conjunto de regras, que definem como utilizar o *XML* para representação dos dados e convenções para representar *remote procedure call (RPCs)*. Noutras palavras, os serviços *web* utilizam mensagens *XML*, codificadas em *SOAP*, para comunicarem.

O *SOAP*, foi desenhado, originalmente, para transportar chamadas de métodos remotos, de um sistema local para um sistema remoto. O que diferencia uma arquitectura baseada no *SOAP* de outras arquitecturas remotas (como o *DCOM* e *CORBA*, por exemplo), é que o protocolo *SOAP* consegue penetrar em quase todos os *firewall* corporativos, e o pacote *SOAP* contém dados codificados em *XML*, que são de fácil descodificação e utilização.

O mundo da computação distribuída é maioritariamente composto de transações e mensagens, e enquanto se utiliza (normalmente) *COM/DCOM*, *CORBA* e *EJBs* para esse efeitos, as aplicações *web* de hoje são normalmente feitas artesanalmente ou utilizam mecanismos complexos para comunicarem entre os diferentes campos tecnológicos.

O *SOAP*, não será certamente a melhor solução para todas as aplicações. Deverá-se deixar sempre as portas abertas ao *COM* e *RMI*, por exemplo.

Uma das perguntas que poderá surgir é se o programador terá de aprender *SOAP* ou *XML*? A resposta é não. O serviço *web* toma conta de toda a codificação e descodificação necessárias. O programador só terá de se preocupar na programação. Utilizar mensagens formatadas com o *SOAP* e o *XML* sobre o *HTTP*, permite aos consumidores do serviço *web* estarem em qualquer plataforma, modelo de objectos e linguagem de programação, desde que o cliente possa interpretar e gerar mensagens *XML* codificadas com o *SOAP*. Isto é a chave para que os clientes dos serviços *web* possam ser inter-operáveis.

4.4 Os objectivos do *Framework .Net*

O *Framework .Net*, é uma culminação dos esforços reunidos de variadas equipas da *Microsoft*, que trabalham juntas para criar uma plataforma de construção e despacho rápido, de aplicações e serviços *web*. A visão por detrás da plataforma do *Framework .Net* é combinar um paradigma de programação de fácil utilização com os protocolos escalonáveis e abertos da Internet. Para atingir esta visão, vários objectos intermediários têm de ser atingidos:

- Integração através de standards públicos da Internet.
As soluções de programação, para comunicarem com parceiros de negócios, clientes, secções separadas geográficamente e até futuras aplicações, precisam de oferecer suporte para os standards abertos da Internet, e de uma integração profunda e transparente com esses mesmos protocolos, que não obrigue o programador a estudar toda a infra-estrutura que está pôr detrás.
- Escalonabilidade através de uma arquitectura “fracamente ligada”.
O *Framework .Net* foi construído para juntar os benefícios da produtividade das arquitecturas “fortemente ligadas”, com as vantagens da escalonabilidade e interoperabilidade das arquitecturas “fracamente ligadas”.
- Suporte multi-língua.
Em vez de obrigar todos a aprender uma única linguagem de programação, o *Framework .Net*, permite que aplicações escritas em diversas linguagens de programação, se integrem umas com as outras. Como resultado, o *Framework .Net* permite às empresas, beneficiar dos conhecimentos de programação actuais sem qualquer necessidade para nova formação, e permite aos programadores, programar na sua linguagem de eleição.
- Aumentar a produtividade dos programadores.
Com a falta mundial de programadores que existe hoje em dia, cada hora de programação tem de ser tornada em trabalho produtivo. A equipa de desenvolvimento do *Framework .Net*, focou a sua atenção em eliminar muita da programação de “canalização”, deixando assim o programador livre para se preocupar unicamente em escrever a “lógica do negócio”.
- Protecção de investimentos através de segurança avançada.
Uma das maiores preocupações com a Internet, hoje em dia, é a segurança e não é necessário dizer, que qualquer solução de programação para a Internet, necessita de ter implementada segurança como componente integral, e não como um componente que é colocado posteriormente. A arquitectura de segurança do *Framework .Net* foi desenhada de base, para garantir que a informação e aplicação estão protegidos através de um modelo de segurança refinado e baseado em factos.

- Utilizar serviços do sistema operativo.

O *Windows* providencia a plataforma mais rica de serviços. O *Framework .Net*, toma partido disso e expõe-nos de tal maneira, para permitir uma utilização fácil.

O *Framework .Net* simplifica o desenvolvimento dos serviços e aplicações providenciando características para ajudar os programadores da *web* a utilizar os serviços *web*, como se estes fossem objectos locais, na linguagem de eleição do programador. Os programadores assim irão focar os seus esforços unicamente nos serviços que dão às suas empresas vantagens competitivas. O resultado é a colocação do produto no mercado mais rapidamente, produtividade do programador aumentada e software de qualidade mais alta.

É evidente que é necessário uma infra-estrutura razoável, para fazer com que a construção de serviços *web* seja transparente para os programadores e utilizadores. O *Framework .Net* da *Microsoft* oferece essa infra-estrutura.

4.5 As três partes do Framework .Net

O objectivo principal do *Framework .Net* é tornar fácil a construção de serviços *web* e aplicações, mas também tem um efeito dramático em todos os tipos de aplicações, desde simples aplicações clientes a muitas outras aplicações distribuídas.

O *Framework .Net* consiste de três partes principais: a *common language runtime*, um conjunto hierárquico de livrarias de classes unificadas, e uma versão componentizada de páginas *Active Server*, chamada *ASP .Net*.

A *common language runtime* está sobre os serviços do sistema operativo. Na realidade, é a responsável por colocar a aplicação a executar – verificando que todas as dependências da aplicação são fornecidas, administrando a memória, administrando a segurança, a integração da linguagem e por ai fora. A *runtime* proporciona muitos serviços que ajudam a simplificar o desenvolvimento do código e da distribuição da aplicação, enquanto melhora a segurança da aplicação.

Contudo, o programador não interage na realidade com a *runtime*. Os programadores utilizam um conjunto de classes unificadas construídas sobre a *runtime*. Estas classes podem ser utilizadas a partir de qualquer linguagem de programação.

O *Framework .Net* inclui um modelo de programação de aplicações *web*, conhecido por *ASP .Net*, como parte desta livrarias. Ele proporciona componentes e serviços de alto nível, especialmente produzidos para desenvolver serviços *web* e aplicações.

4.5.1 A Common Language Runtime

A *common language runtime* é uma máquina de execução de alto nível. O código que tem como objectivo a *runtime*, e cuja execução é manuseada pela *runtime*, é referido como código manuseado. A responsabilidade para tarefas como criar objectos é delegada á *common language runtime*, que permite á *runtime* proporcionar serviços adicionais ao código a ser executado.

Apesar do seu nome (em inglês), a *common language runtime*, tem na realidade um papel, quer no tempo de desenvolvimento do componente, quer no tempo de execução de experiências.

Enquanto o componente está a ser executado, a *runtime* proporciona serviços tais como administração de memória (incluindo a colecção de lixo – *garbage collection*), administração do processo, administração dos *threads*, cumprimento de segurança, bem como satisfazer quaisquer dependências que o componente possa ter noutras componentes.

Em tempo de desenvolvimento, o papel da *runtime* muda um pouco. Devido a todas as automações que torna possível, a *runtime* torna a experiência do programador muito simples. Muitas das suas características reduzem a quantidade de código que um programador tem de escrever, para tornar a “lógica do negócio” em componentes reutilizáveis.

As *runtimes* não são novidade para as linguagens: praticamente todas as linguagens têm uma *runtime*. O papel critico da *common language runtime*, é o que na realidade a torna diferente. Ela proporciona um ambiente de *runtime* unificado para todas as linguagens de programação.



Figura 4.2 – A commom language runtime

4.5.1.1 As principais características

As principais características da *runtime*, incluem um sistema de tipos comuns, componentes auto-descritivos, distribuição simplificada e serviços de segurança integrados.

- O sistema de tipos comuns e a integração multi-linguagem.
A *runtime* faz uso do novo sistema de tipos comum, capaz de exprimir a semântica das linguagens de programação modernas. O sistema de tipos comuns define um conjunto de tipos de dados e regras standard para criar novos tipos. A *runtime* comprehende como criar e executar este tipos. Os

compiladores utilizam os serviços *runtime* para definir tipos de dados, manusear objectos, e fazer chamada a métodos, em vez de utilizar métodos específicos de ferramentas ou linguagens.

O resultado de ter um sistema de dados comum, é uma integração multi-linguagem profunda. O código escrito numa linguagem pode herdar a implementação de classes escritas noutra linguagem. Isto quer dizer que os programadores já não precisam de criar versões diferentes da suas bibliotecas reutilizáveis para cada linguagem de programação ou compilador, e os programadores que utilizam as livrarias de classes, deixam de estar limitados ás bibliotecas desenvolvidas para as linguagens de programação que estão a utilizar.

- *Metadata* e componentes auto-descritivos.

O *Framework .Net* permite a criação de componentes auto-descritivos, o que simplifica o desenvolvimento e a distribuição, e melhora a segurança do sistema. A auto-descrição é conseguido através da *metadata* – informação contida no binário que suplementa o código executável, proporcionando detalhes acerca de dependências, versões, etc.. A *metadata* é “empacotada” juntamente com o componente que descreve, resultando assim em componentes auto-descritivos.

Uma das principais vantagens dos componentes auto-descritivos, é o facto de não ser necessário mais nenhum ficheiro para se poder utilizar o componente. Isto é um contraste face ás aplicações de desenvolvimento típicas actuais, que necessitam de ficheiros separados de cabeçalho para a definição das classes, de ficheiros (separados) da linguagem de descrição da interface e livrarias de tipos de dados separadas e *proxies* separadas. Uma vez que a *metadata* é gerada do código de origem durante o processo de compilação, e é guardada com o código executável, nunca esta desfaçada do executável.

Para além de resolver os desafios de desenvolvimento, a auto-descrição reduz a dependência do *Registry* do *Windows*. Devido ao facto de que cada aplicação contém uma descrição completa de si própria, a *runtime* pode construir dinamicamente uma *cache* de informação sobre o componente instalado no sistema. Se essa *cache* eventualmente se estragar, a *runtime* pode, por exemplo, reconstruí-la sem que o utilizador saiba sequer que o fez.

- Distribuição “não-toca”.

O *Framework .Net* inclui avanços que eliminam quase por completo o inferno das *DLLs* – este é o fenómeno no qual um biblioteca distribuída fica dessincronizada das aplicações que lhe estão a tentar aceder.

Em primeiro lugar, o *Framework .Net* tem incluído um sistema de atribuição de nomes interno bastante forte o que torna a troca entre duas livrarias com o mesmo nome, bastante difícil. Para além disso, existe uma característica conhecida por distribuição “lado-a-lado”. Se uma nova aplicação sobre-escrever uma livraria distribuída, uma aplcação existente que não pode utilizar a nova livraria distribuída, pode “arranjar-se” sozinha. Da próxima vez que a aplicação existente arrancar, vai verificar os seus ficheiros distribuídos. Se descobrir que um deles mudou e que as alterações são incompatíveis, ela pede á *runtime* para ir buscar uma versão que ela sabe que vai funcionar. Devido a este sistema de segurança, a *runtime* pode faze-lo em segurança, e a aplicação pode corrigir-se sozinha.

- Segurança baseada em factos.

O *Framework .Net* deu uns grandes passos em frente no que respeita á segurança, introduzindo um sistema de segurança baseada em factos bastante refinado. Este sistema de segurança proporciona ao programador e administrador, um grande leque de privilégios, os quais eles podem conceder (não é só *on* ou *off*). Mais, permite que esses privilégios possam ser aplicados baseados nos aspectos gerais do próprio código, incluindo a origem do código e as assinaturas digitais.

Para além disto, uma vez que a *common language runtime* é utilizada para “carregar” código, criar objectos e fazer chamadas a métodos, a *runtime* pode na realidade fazer verificações de segurança de baixo nível e enforcar a política de segurança conforme o código é “carregado” e executado.

Este plano garante que utilizadores não autorizados, não podem aceder aos recursos e o código não pode realizar acções não autorizadas, o que na geral melhora a segurança e confiança do sistema.

4.5.2 As Classes Unificadas (o que o programador vê)

As classes do *Framework .Net* providenciam um conjunto de livrarias de classes unificadas, orientadas a objecto, hierárquicas, e extensíveis (*APIs*), que os programadores podem utilizar nas linguagens de programação com que já estão familiarizados.

Hoje em dia, os programadores que utilizam o *Visual C++*, utilizam as *Microsoft Foundation Classes*, os programadores que utilizam o sistema de desenvolvimento *Visual J++* utilizam as *Windows Foundation Classes*, e os programadores que utilizam o *Visual Basic* utilizam o *Visual Basic framework*. Ou seja, as classes do *Framework .Net* unem estas classes diferentes, criando um super-conjunto com as suas características. O resultado é que os programadores já não necessitam de aprender os modelos múltiplo objectos ou as livrarias de classes. Criando um conjunto comum de *APIs* que abrange todas as linguagens de programação, o *Framework .Net* permite herança através de qualquer linguagem, tratamento de erros e o *debug*. De facto, todas as linguagens de programação, desde *Jscript* até *C++*, tornam-se iguais, e os programadores estão assim livres de escolher a língua correcta para o serviço a ser executado.

O *Framework .Net* providencia classes que podem ser chamadas de qualquer linguagem de programação. Estas classes obedecem a um conjunto de linhas guia de atribuição de nomes e de desenho para diminuir ainda mais a curva de aprendizagem do programador. Algumas das livrarias de classes chave, estão ilustradas na figura abaixo.



Figura 4.3 – As classes unificadas do Framework .Net

4.5.3 ASP .Net – Aplicações Web

ASP .Net, um dos conjuntos de classes dentro da livraria das classes unificadas, proporciona um modelo de aplicação web na forma de um conjunto de controlos e uma infra-estrutura que torna fácil construir aplicações web.



Figura 4.4 – Uma visão da ASP .Net

A *ASP .Net* vem com um conjunto de controlos para o servidor (às vezes são chamados *forms web*) que são muito idênticas aos *widgits* típicos da interface de utilizador *HTML* (inclui listas, caixas de texto e botões) e um conjunto adicional de *controls web* que são mais complexos (tais como calendários). Estes *controls* são na realidade executados no servidor *web* e projectam a sua interface de utilizador como *HTML* para um *browser*. No servidor, os *controls* expõem um modelo de programação orientado a objecto, que trás a riqueza da programação orientada a objecto ao programador *web*.

Uma característica importante destes *controls* é que eles podem ser desenvolvidos para se adaptarem a capacidades do lado do utilizador; as mesmas páginas podem ser utilizadas para terem em mira um leque vasto de plataformas de cliente e factores de *form*. Ou seja, os *controls* das *forms web* conseguem “cheirar” o cliente que está a requisitar a página e retornam uma experiência de utilizador apropriada – *WML* para

telefones ou *HTML 3.2* para *software de browser* de baixo nível e *HTML dinâmico* para o *Internet Explorer 5.5*. Esta característica é referida como “interface de utilizador adaptável”.

A *ASP .Net* utiliza estes mesmos conceitos para permitir aos programadores entregar o software como um serviço. Utilizando as características dos serviços *web ASP .Net*, os programadores *ASP .Net* podem simplesmente escrever a sua “lógica de negócio” e a infra-estrutura *ASP .Net* responsabiliza-se a entregar esse serviço via *SOAP* e outros protocolos publico.

A *ASP .Net* funciona com todas as linguagens de programação e ferramentas.

4.5.3.1 Dentro da *ASP .Net*

No centro da *ASP .Net* esta a *runtime HTTP* (que é diferente da *commom language runtime*), uma máquina de execução de alto nível para processamento de comandos *HTTP*. A *runtime HTTP* é responsável pelo processamento de todos os pedidos *HTTP* que chegam, resolver o *URL* de cada pedido para uma aplicação, e depois enviar o pedido á aplicação para a continuação do processamento. A *runtime HTTP* é *multithreaded* e processa os pedidos assincronamente, o que quer dizer que não pode ser bloqueada para processar novos pedidos, pelo mau código de uma aplicação. Mais, a *runtime HTTP* tem um desenho resiliente, por isso está preparada para recuperar automaticamente de violações de acesso, perdas de memória, *deadlocks*, e por ai fora.

Actualizar aplicações. A *ASP .Net* utiliza as tecnologias de distribuição do *Framework .Net*, e assim ganha todos os seus benefícios tais como o envio *XCOPY* e a distribuição “lado-a-lado” das aplicações.

Um outro dos principais benefícios da *ASP .Net* é o suporte de actualizações de aplicações “em directo”. Um administrador não necessita de desligar o servidor *web* ou mesmo a aplicação para actualizar os ficheiros da aplicação, os ficheiros da aplicação nunca estão “trancados” (*locked*), por isso podem ser sobreescritos mesmo quando a aplicação está a ser executada. Quando os ficheiros são actualizados, o sistema “elegantemente” muda para a nova versão.

Pipeline extensível. Dentro de uma aplicação, os pedidos *HTTP* são dirigidos por um *pipeline* de módulos *HTTP*, para um “tratador” de pedidos. Os módulos *HTTP* e o “tratador” de pedidos, são simplesmente coordenados pelas classes *.Net* que implementam interfaces específicas definidas pela *ASP .Net*. A arquitectura *pipeline* torna muito fácil adicionar serviços ás aplicações; basta providenciar um modulo *HTTP*. Por exemplo, a segurança, manuseamentos de estados e a tracagem estão implementadas como módulos *HTTP*. Modelos de programação de alto nível, tais como serviços *web* e forms *web*, são geralmente implementados como “tratadores” de pedidos.

Uma aplicação pode ser associada a múltiplos “tratadores” de pedidos – um por *URL* – mas todos os pedidos *HTTP* são direcionados pela mesma *pipeline*.

Manuseamentos de estados. A *web* é fundamentalmente um modelo sem estado, sem correlação entre pedidos *HTTP*. Isto pode tornar o desenvolvimento de aplicações *web* difícil, visto que as aplicações normalmente necessitam de manter estados dos múltiplos pedidos. A *ASP .Net* melhora os serviços de manuseamento de estado, introduzo pela *ASP*, para proporcionar três tipos de estados para as aplicações *web*: aplicação, sessão e utilizador.

O estado de sessão da *ASP .Net* é guardada num processo separado e pode até ser configurado para ser guardado numa máquina diferente. Isto torna o estado de sessão utilizável, quando uma aplicação é distribuída numa *form web*.

O estado de utilizador é semelhante ao estado de sessão, mas geralmente não faz *time out* e é persistente. Assim o estado de utilizador é útil para guardar preferências do utilizador e outras informações de personalização. Todos os serviços de manuseamento dos estados são implementados como módulos *HTTP*, para que possam ser adicionados ou removidos facilmente do *pipeline* de uma aplicação. Se forem necessários mais serviços de manuseamento de estados, para além daqueles providenciados pela *ASP .Net*, podem ser providenciados por terceiros.

Caching. A *ASP .Net* providencia serviços de *caching* para melhorar as “actuações”. Uma *cache* de *output* guarda por completo páginas que foram referidas, e uma *cache* de fragmentos guarda páginas parciais. As classes são providenciadas para que aplicações, módulos *HTTP*, e “tratadores” de pedidos, possam guardar objectos arbitrários na sua *cache* conforme necessitem.

Capacidades de *caching* agressivas irão ser proporcionadas como parte do modelo de programação da *ASP .Net*. Está desenhado para proporcionar um ambiente de aplicação *web* robusto, capaz de executar projectos de missão crítica por longos períodos de tempo.

Embora a *ASP .Net* não esteja disponível no mercado, já está a proporcionar melhorias significantes nas aplicações *web* – melhoria nas *performances* de 3x mais nas aplicações baseadas em *ASP* existentes, e melhorias de produtividade ainda mais dramáticas.

4.6 Produtividade do programador

Integração multi-linguagem

Os objectos escritos por cima do *Framework .Net*, podem-se integrar uns com os outros indiferentes da linguagem em que foram criados porque estão construídos sobre um sistema de tipos comuns. A integração multi-linguagem do *Framework .Net* inclui:

- chamada de métodos de outros objectos;
- herdar implementações de outros objectos;
- passar instâncias de uma classe para outros objectos;
- utilizar um único *debugger* para múltiplos objectos e
- “apanhar” erros de outros objectos.

Existem inúmeros cenários onde a integração de linguagens é útil. Por exemplo, projectos de desenvolvimento têm acesso a uma base de aptidões maior – eles podem escolher de programadores aptos em qualquer linguagem de programação e juntarem-se a eles, na mesma equipa. Alternadamente, se nós estivermos a escrever componentes para uma aplicação *web* distribuída, seria bom saber que qualquer linguagem que escolher-mos para escrever o nosso componente, os componentes podem interagir uns com os outros e com os componentes disponibilizados por outros programadores.

Escrever menos código

O *Framework .Net* diminui a quantidade de código que o programador tem de escrever. Isto é feito através de muitos mecanismos tais como *metadata* auto-descritiva, administração de memória automática, um conjunto rico de *controls*, classes de programação simplificadas, e a capacidade de transformar automaticamente qualquer método de aplicação num serviço *web*, com a simples adição de uma palavra chave.

Uma aplicação para o mundo: localização

O *Framework .Net* permite aos programadores criar uma aplicação ou um *site web* para o mundo inteiro, independentemente do utilizador final. O *Framework .Net* faz isto utilizando dados *Unicode* e oferecendo um administrador de pedidos, integrado no *Framework .Net* para permitir às aplicações que troquem facilmente a linguagem de interface do utilizador.

O *Framework .Net* foi desenhado do zero para manusear os dados internos com *Unicode/UTF-8*. Isto tem três implicações: os componentes construídos sobre o *Framework .Net* podem manusear dados como *Unicode* ou *UTF-8*; os programadores podem proporcionar uma experiência de utilizador multinacional, indiferente da plataforma de desenvolvimento (por exemplo, *Windows 2000*, *Windows NT*, *Windows 98*, etc.); um desenvolvimento internacional de uma aplicação não é dependente de uma versão especial do *runtime*. Se um programador estiver a utilizar a versão japonesa do *Framework .Net*, ele pode também escrever aplicações para consumidores ingleses ou portugueses.

O *Framework .Net* também providencia versões culturais de funcionalidades comuns, tais como ordenamento e formatação de data e hora.

Criar serviços *web* com facilidade

Tirar partido da Internet utilizando interfaces de componentes *COM*, foi tornado simples em 1996 com o lançamento dos *scripts* das páginas *Active Server (ASP)* a correr sobre o *Windows 2000 Server's Internet Information Services (IIS)*. A *Microsoft* está a tornar fazer isto ainda mais facilmente e com maior confiança. Com o *Framework .Net*, todos os objectos podem-se tornar acessíveis á Internet, marcando-os simplesmente com a palavra chave: *WebMethod*.

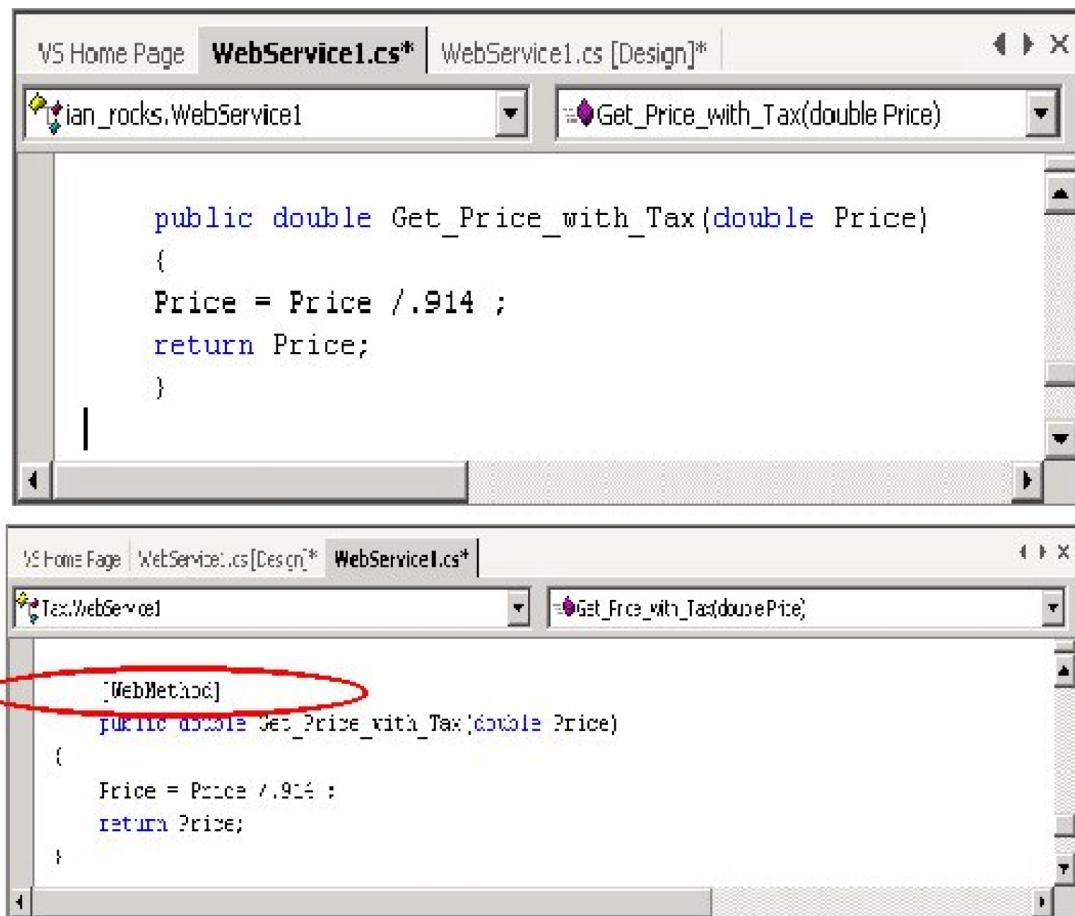


Figura 4.5 – Uma função em C# que se torna disponível como um serviço web

Um pormenor importante, é que no primeiro exemplo temos uma função que não estava disponível na Internet, até a palavra chave *[WebMethod]* ter sido introduzida.

Mas isto é somente metade da solução; a palavra chave *WebMethod* dá aos programadores a capacidade de exporem funcionalidades na Internet, mas como é que um outro programador utiliza estas mesmas funcionalidades? O Framework .Net tem uma ferramenta que é o *WebServiceUtil.exe* que leva o caminho para um ficheiro *WSDL* e uma linguagem (*C#, Visual Basic* ou *JavaScript*), como argumentos de *input*. A ferramenta gera então um único ficheiro de origem como *output*. Este ficheiro contém a classe *proxy*, define a linguagem especificada, que expõe métodos (ambos síncronos e assíncronos) para cada um dos métodos expostos pelo serviço web. Cada método *proxy* contém a invocação de rede apropriada e a ordenação de código necessária para invocar e receber uma resposta dos serviços remotos. O *Visual Studio .Net* (do qual irei falar mais adiante), torna isto ainda mais fácil fazendo-o através de uma interface de exploração.

Utilização de serviços COM+

Introduzido em 1992, o *Microsoft Component Object Model (COM)* é o modelo de objecto mais popular para construir componentes reutilizáveis, e têm mais de 10.000 componentes disponíveis comercialmente. Com o tempo, o *COM* desenvolveu de uma tecnologia de conter um documento dentro de outro (*OLE*), para uma tecnologia, para

distribuir aplicações contidas em si próprias (*Active X*), para uma tecnologia para construir e correr “lógicas de negócio” multi-cliente (*COM+*).

Com tantos anos de experiência, os arquitectos da *Microsoft* quiseram tornar o desenvolvimento *COM* mais fácil. O *Visual Basic* começou por mostrar como se poderia fazer isto em 1995. O *Framework .Net* leva essa simplificação um passo mais á frente, automatizando características do *COM* como a contagem de referencias, descrição da interface e registro.

O *Framework .Net* suporta o *COM* nativamente. De facto, um programador de *COM* que utilize o *Visual Studio 6.0* poderia chamar um componente *Framework .Net* e para o programador iria parecer um componente *COM*, completo com *iUnknown*. Da mesma forma, um programador *Framework .Net* a utilizar o *Visual Studio .Net* iria ver um componente *COM*, como um componente *Framework .Net*.

COM+, é *COM* combinado com *MTS*, *DCOM*, e outros serviços de componentes tais como *Partitions*. O *COM+* providencia um conjunto de serviços orientados a multi-utilizadores. Em particular, *COM+* providencia administração de processos. Em versões futuras, irá também providenciar isolação de processos mais forte, desenhado para os *service providers* de aplicações - uma característica conhecida por *partitioning*.

Os serviços *COM+* estão complementados aos serviços de programação proporcionados pelo *Framework .Net*. O *Framework .Net* providencia acesso directo a estes serviços de componentes.

Uma das características mais cruciais do *COM+*, que os programadores vão querer utilizar do *Framework .Net*, são as transacções *COM+*. É fácil construir aplicações que necessitam suporte de transacção através de palavras chaves simples e *settings*, no *Framework .Net*. O *Framework .Net* tira de facto partido das vantagens da infra-estrutura das transacções *COM+* para proporcionar transacções distribuídas manuais e automáticas.

A beleza da integração entre o *Framework .Net* e as transacções *COM+* é que os programadores *Framework .Net* podem utilizar o ambiente de transacção *COM+* de alto nível sem precisar de aprender *COM+*. Por exemplo, podemos copiar simplesmente componentes *Framework .Net* que necessitam serviços *COM+* tais como transacções numa directória, e o *Framework .Net* irá adiciona-los dinamicamente ao monitor de transacções *COM+*.

5 Visual Studio .Net

A visão da *Microsoft* para o *Visual Studio .Net*, é de simplificar o desenvolvimento de serviços *web* e aplicações distribuídas. Tal como os programadores evoluíram da construção de aplicações monoposto *Windows*, para a construção de aplicações cliente/servidor, o *Windows* e o *Visual Studio* evoluíram para os ajudar. Agora que as empresas estão de caras com o desafio de integrarem clientes, parceiros e fornecedores, os programadores tem de focar a sua atenção para aplicações e serviços *web* distribuídos. O *Visual Studio .Net* irá proporcionar produtividade e poder topo da gama para cada aspecto dessas aplicações.

Hoje em dia, existem mais de 6 milhões de programadores profissionais em todo o mundo, dos quais mais de 70% (**diz** a *Microsoft*) utilizam ferramentas da *Microsoft* para construir as suas aplicações. Enquanto a arquitectura de aplicação, mais popular é a do cliente/servidor, a arquitectura nova que cresce mais depressa é a *web*, com um *browser* para interacção do utilizador e uma lógica de programação e/ou com conexões para a base de dados no servidor.

A *Microsoft* tem permitido o desenvolvimento de aplicações *web* com os dois lançamentos do *Visual Studio* prévios, bem como através das páginas *Active Server (ASP)*, uma tecnologia de desenvolvimento *web* que foi primariamente distribuída com o *IIS 3.3*. Para continuar a endereçar a necessidade deste segmento de crescimento tão rápido de programadores, a *Microsoft* tem conduzido centenas de reuniões, estudos, e tem revisto as opiniões de milhares de programadores.

O *Visual Studio .Net* planeia ser um *framework* de linguagem aberta e um ambiente de desenvolvimento para suportar mais de uma dúzia de linguagens, e as livrarias comuns que o *CLR* planeia providenciar deverá permitir uma fácil chamada e *debug* de código numa linguagem, noutra qualquer linguagem.

O *Visual Studio .Net* tem todos os requisitos para a próxima grande evolução no desenvolvimento de aplicações, simplificando dramaticamente o desenvolvimento de aplicações e serviços *web* distribuídos.. O *Visual Studio .Net* tira partido, por completo, do *Framework .Net*, que utiliza standards públicos da Internet para permitir uma integração com aplicações novas e existentes, que correm numa qualquer plataforma.

5.1 Os objectivos

Quando a equipa, que iria produzir o *Visual Studio .Net*, começou a planeia-o eles tentaram guiarem-se por um conjunto de objectivos. Os objectivos eram:

- Melhorar a produtividade do programador.
- Ferramentas poderosas para o desenvolvimento em *Windows* e na *web*.

O *Visual Studio .Net* irá proporcionar o melhor conjunto de ferramentas para os programadores construir um leque mais abrangente de aplicações.

- Inovações nas linguagens.

A maioria dos programadores profissionais utilizam mais que uma linguagem.

O *Visual Studio .Net* renovou a linguagem de desenvolvimento mais popular do mundo, o *Visual Basic*, proporciona melhoramentos importantes na

poderosa linguagem *C++*, e introduz uma linguagem nova, moderna e orientada ao objecto, o *C#*.

- Simplificar o desenvolvimento baseado no servidor.

Uma das tarefas mais complexas no desenvolvimento de aplicações distribuídas é a construção de componentes para o servidor (para o lado do servidor) que implementem o *business logic* (a “lógica do negócio”) de uma aplicação. O *Visual Studio .Net* facilita esta construção.

- Simplificar a criação e utilização de serviços *web*.

Nesta próxima geração da Internet, onde os sites *web* se tornam em serviços *web* programáveis, os programadores necessitam de poder expor facilmente qualquer aplicação como um serviço *web* e de conseguirem sem grandes dificuldades consumir um serviço *web* nas suas aplicações.

5.2 As características

Uma vez vistos os principais objectivos com que foi elaborado o *Visual Studio .Net*, vamos ver agora o que as principais características, permitem aos programadores.

- Tornarem-se mais rápidos com as ferramentas de desenvolvimento mais produtivas.

Um ambiente visual bastante integrado e unificado, simplifica o processo de desenvolver aplicações *web* e diminui a curva da aprendizagem significativamente. O *HTML*, *XML* e os editores de *style-sheet* distribuídos tornam fácil desenvolver aplicações *web* a partir de qualquer linguagem do *Visual Studio*, incluindo a linguagem de programação *C#*.

- Desenhar rapidamente aplicações *web* mais abrangentes.

Com as *forms web*, os programadores podem criar aplicações *web* programáveis indiferentes à plataforma e *browser*, utilizando as mesmas técnicas que eram utilizadas para construir aplicações desktop, baseadas em *forms*. As *forms web* são executadas no servidor *web*, permitem uma execução rápida em *runtime* e produzem documentos complacentes com *HTML 3.2* que “correm” em qualquer *browser*.

- Utilizar serviços *web XML* de uma forma fácil para simplificar a computação distribuída.

Com os serviços *web XML* os programadores podem invocar o *business logic* (a lógica do negócio) utilizando os protocolos de Internet standards. Os serviços *web XML* utilizam *HTTP* como transporte. *XML* é utilizado para formatar os parâmetros do pedido de *input* e *output*, para que o pedido não fique dependente de nenhum componente tecnológico ou convenção de chamada de objectos. O resultado é, serviços *web XML* que podem ser acedidos por qualquer linguagem, utilizados por qualquer modelo de componentes, e executam em qualquer sistema operativo.

- Construir soluções escalonáveis e disponíveis.

5.3 As ferramentas da produtividade

O ambiente de negócios dinâmicos de hoje em dia, exige que as aplicações sejam concluídas rapidamente, sejam mantidas com facilidade, e frequentemente melhoradas

(*upgrade*). Com este cenário, maximizar a produtividade do programador é um componente crucial para o sucesso.

O *Visual Studio .Net* aborda esta necessidade proporcionando uma série de ferramentas de produtividade e melhorias, que simplificam dramaticamente a criação de aplicações para a *web* bem como para o *Windows*.

Algumas dessas ferramentas de produtividade são:

- O novo ambiente de desenvolvimento integrado distribuído (*IDE – Integrated Development Environment*).
O *Visual Studio .Net* providencia um novo e unificado *IDE* para todas as suas linguagens. Foi desenhado para ajudar os programadores a construir as suas soluções mais rapidamente, com menos confusão e com todas as ferramentas facilmente acessíveis em qualquer linguagem do *Visual Studio*. Mais, o *IDE* unificado, simplifica em muito uma tarefa com que muitas casas de desenvolvimento se deparam hoje em dia – manuseamento e entrega de soluções multi-linguagem. Este ambiente, combina os melhores elementos dos *IDEs* de ferramentas individuais (*Visual Basic*, *Visual C++*, etc.), com novas características integradas tais como o *debugger end-to-end* para proporcionar aos programadores um ambiente de desenvolvimento mais consistente e completamente costumizável.
- A nova página *Start* do *Visual Studio*.
Cada vez que um programador inicia o *Visual Studio*, a página *Start* é exibida por defeito, e é a *homepage*, por defeito, do *browser web* para o *IDE*. Proporciona uma localização central para definir as opções preferidas, ler as notícias dos produtos, aceder a discussões com outros programadores, e aceder a outras informações para pôr a funcionar o ambiente *Visual Studio .Net*. Mais do que providenciar acesso instantâneo a artigos, eventos, e tópicos de ajuda do *MSDN Online*, a página *Start* permite aos programadores aceder a projectos recentes, existentes ou novos com o *click* de um botão. A página *Start* também permite aos programadores, adaptarem-se rapidamente à aparência do *IDE*, para melhor se enquadrar com as suas experiências de programador.

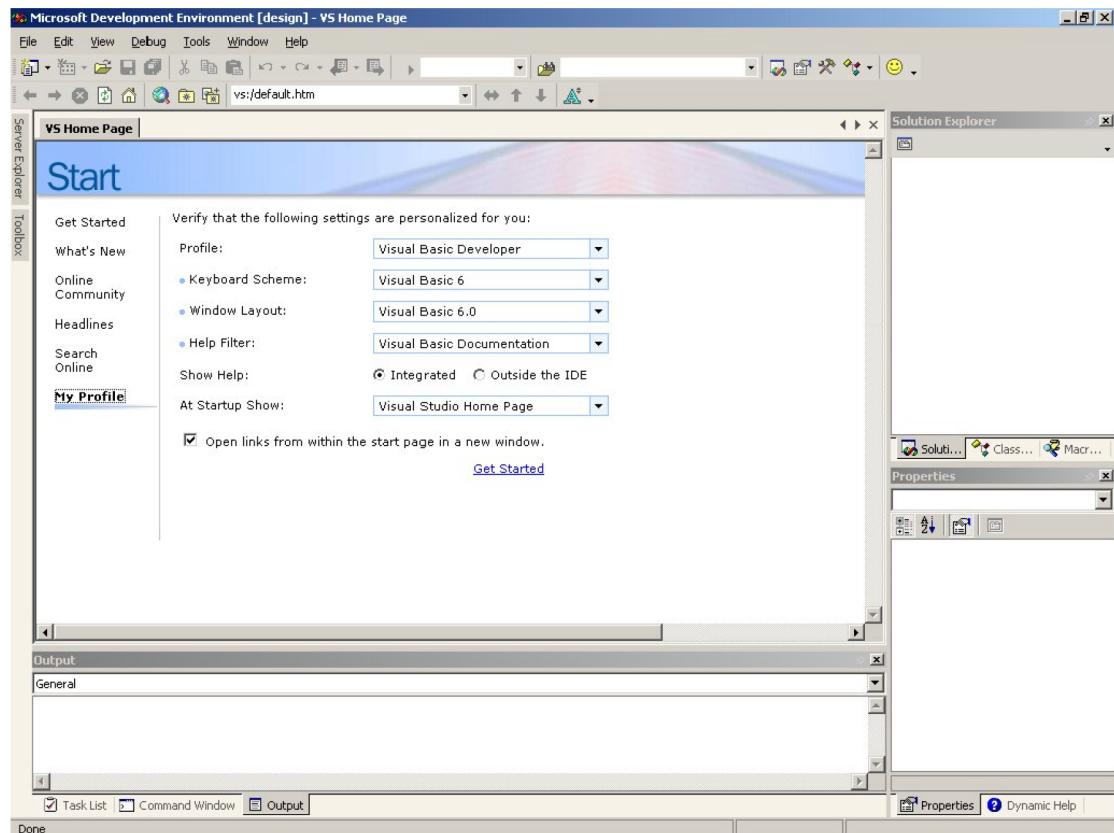


Figura 5.1 – A página Start do Visual Studio

- Um *Solution Explorer* melhorado.
- No *Visual Studio .Net*, o conceito do explorador de projecto foi extendido para um melhorado e multi-linguagem *Solution Explorer*. O *Solution Explorer* mostra uma lista organizada dos projectos bem como os seus ficheiros e directórios correspondentes, que fazem parte da solução corrente. Proporciona informação em tempo-real sobre itens na solução aberta e permite um manuseamento destes itens aos programadores.

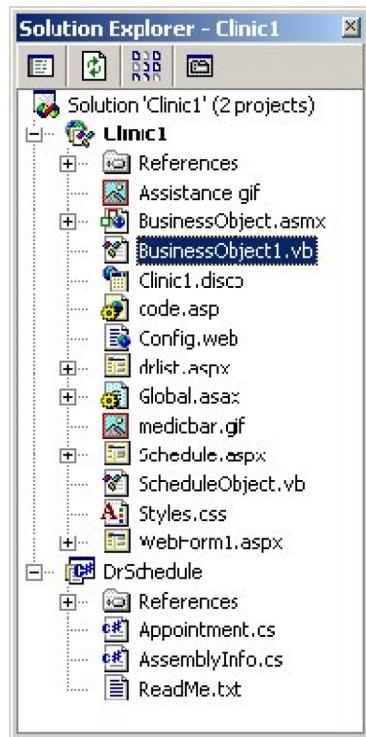


Figura 5.2 – O *Solution Explorer* mostra a hierarquia de todos os projectos e ficheiros correspondentes, de uma dada solução

- A *Toolbox* (caixa de ferramentas) melhorada.

Os programadores do *Visual Studio 6.0* vão ficar familiarizados imediatamente com a *Toolbox* do *Visual Studio .Net*. Esta janela de *Toolbox* melhorada mostra uma variedade de itens para serem utilizados nos projectos *Visual Studio .Net*, incluindo *forms web* e *controls* de *forms Windows*, *controls ActiveX*, serviços *web*, elementos *HTML*, objectos, e ficheiros de texto. Os itens disponíveis nesta janela, mudam de acordo com o *designer* ou editor que o programador está a utilizar na altura.



Figura 5.3 – A janela Toolbox (caixa de ferramentas).

- O novo *Server Explorer*.
O *Server Explorer* é uma nova consola de desenvolvimento para servidores, do *Visual Studio .Net*. É uma janela de ferramenta distribuída que ajuda os programadores a aceder e manipular recursos em qualquer máquina para a qual têm permissões. Com o *Server Explorer*, os programadores podem “ligar-se” a servidores e visualizar os seus recursos baseados em servidores, incluindo filas de mensagens, contadores de performance, serviços, processos, diário de eventos (*event log*) e objectos de base de dados.

Mais, o *Server Explorer* permite que os programadores referenciem programaticamente estes componentes e recursos de servidor com as suas aplicações baseadas no *Visual Studio .Net*, adicionando um componente ao seu projecto que referencia o recurso ou criando componentes que monitorizam a actividade do recurso. Isto inclui, fazer ligações de dados ao *SQL Server* ou a outras bases de dados, configurar ou integrar os servidores *Exchange 2000* na aplicação, monitorizar processos, serviços e *DLLs* carregadas num servidor e fazer o *debug* de eventos do servidor.

Finalmente, o *Server Explorer* dá aos programadores acesso directo a todos os serviços *web* disponíveis num servidor em particular. Com a utilização do *Server Explorer*, os programadores tem a possibilidade de visualizar informação dos métodos e esquemas que os serviços *web* tornam disponíveis,

e podem instantaneamente criar referencias aos serviços web para serem utilizados numa aplicação.

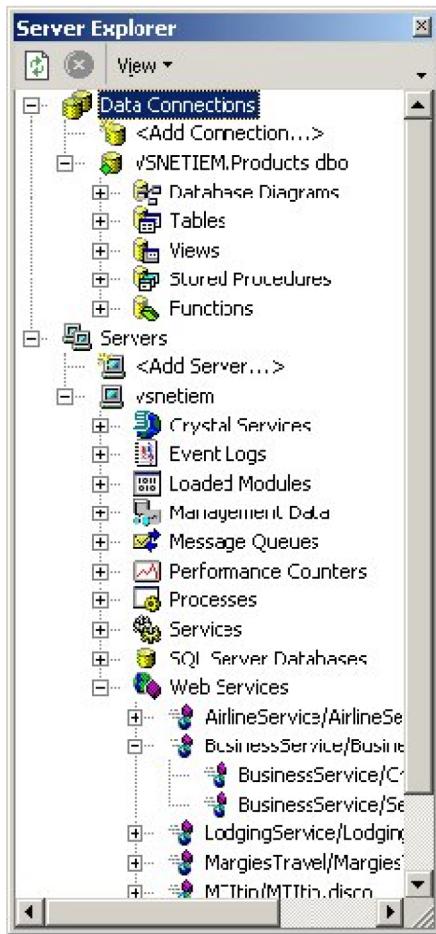


Figura 5.4 – O Server Explorer

- A nova lista de tarefas.

Para além de escrever código e criar componentes que abrangem uma aplicação, os programadores necessitam poder anotar no seu código, para quando eles ou outro membro da equipa o abram mais tarde no ciclo de desenvolvimento, possam determinar, sem atrasos, o estado exacto do código. A lista de tarefas distribuída do *Visual Studio .Net* proporciona aos programadores, esta capacidade crucial permitindo-os marcar o seu código com comentários especializados – estes comentários são então processados e expostos num formato tabular na lista de tarefas. Para além da declaração por defeito *TODO*, os programadores também podem identificar que *tags* a lista de tarefas irá processar.

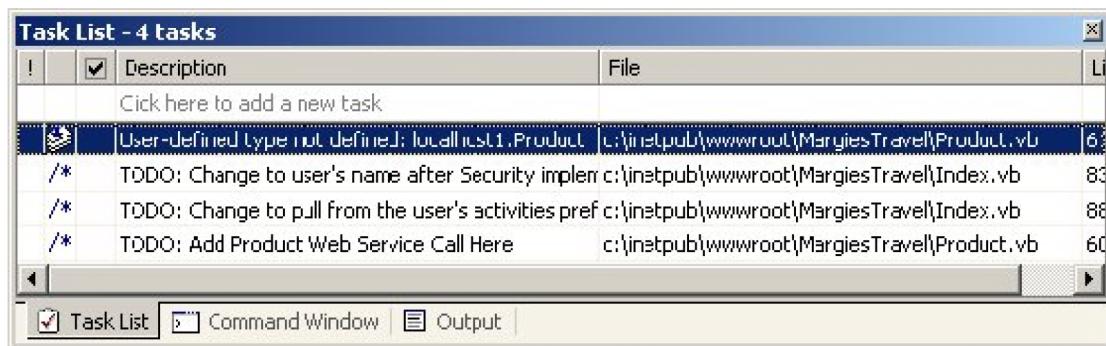


Figura 5.5 – A nova lista de tarefas.

- A nova ajuda dinâmica.

A janela da ajuda dinâmica proporciona o acesso com apenas um *click* á ajuda pertinente, sem se preocupar que tarefa o programador estava atentar completar. A tracagem das seleções que o programador faz, a colocação do cursor, e os itens em foco no *IDE*, fazem com que a ajuda dinâmica filtre os tópicos disponíveis no *MSDN* e proporcionam ponteiros para informação relevante e específica ao desenvolvimento corrente da tarefa que esta a ser feita.

A janela da ajuda dinâmica separa a informação que mostra em diversas categorias, que incluem Ajuda, Ações, Diversos e Exemplos. Os *links* dentro destas categorias são mostradas de acordo com a sua relevância.



Figura 5.6 – A janela da ajuda dinâmica

- O debugger integrado, melhorado.

O *Visual Studio .Net* contém um *debugger* integrado melhorado que diminui o ciclo de desenvolvimento dando aos programadores uma maneira fácil de correr, traçar e corrigir erros no seu código. A interface unificada combina as melhores características do *Visual C++* e dos *debuggers* do *Visual Basic* com

uma cadeia de novas características. Utilizando o *debugger* melhorado, os programadores podem traçar as aplicações *multi-threaded* mais complexas.

- O novo *designer* de *forms web*.

Os programadores de hoje necessitam de uma forma intuitiva de criar página *web*. O *Visual Studio .Net* inclui um *designer* distribuído do *Visual Studio*, de *forms web*. Inclui também, uma maneira gráfica de desenvolver páginas *HTML* e *forms web ASP .Net*, sem a necessidade de um “afundamento” em código *HTML* ou *script*.

Objectos no *designer* de *forms web* são representados na *form*, em directo no modo de desenho. Os programadores podem construir as suas páginas *web* e ver ao mesmo tempo, como iriam ficar, quando estiverem completas.

Os programadores podem também adicionar facilmente elementos *HTML* às suas páginas *web*, arrastando e largando (*drag and drop*) da caixa de ferramentas *HTML* para criarem páginas *web* visualmente complexas, sem ter que sequer escrever uma única linha de código. Os programadores têm assim o benefício do *layout* visual sem as complicações de terem que se lembrar do código *HTML*.

- O novo *designer* de *forms Windows*.

As *forms Windows* são a nova plataforma para o desenvolvimento de aplicações baseadas no *Windows*, que foram baseadas no *Framework .Net*. Elas proporcionam um extenso conjunto de classes concisas e orientadas a objecto que permitem aos programadores desenvolver rapidamente aplicações ricas baseadas no *Windows*.

Utilizando o *designer* das *forms windows*, os programadores podem rapidamente desenvolver soluções para serem utilizadas em aplicações baseadas no *Windows*. Adicionando, simplesmente uma nova *form* a um projecto, o programador tem uma base da qual pode criar rapidamente interfaces de utilizador ricas e intuitivas. Uma vez que uma *form Windows* é adicionada a uma solução *Visual Studio .Net*, o programador pode estabelecer as propriedades da *form*, adicionar *controls* da caixa de ferramentas, e escrever código por detrás da *form*.

Mais, o *designer* das *forms Windows* oferece novas características para permitir aos programadores enviarem as suas soluções ricas, num espaço de tempo muito mais curto. Por exemplo, os programadores podem utilizar a propriedade *Anchor* dos *controls* de uma *form Windows* para se certificarem (com toda a certeza) que a distância entre o *control* e uma das beiras especificadas é sempre constante quando se fizer o *resize* da *form*. Isto diminui o tempo e as complicações que os programadores normalmente tinham para escrever código para fazer o *resize* e o reposicionamento dos *controls*.

- O novo *designer* de componentes.

Da mesma forma que os *designers* das *forms* permitem uma rápida criação de aplicações cliente, o *designer* de componentes proporciona uma maneira de construir rapidamente e graficamente, componentes do lado do servidor. Em vez de muito código, os programadores podem agora arrastar e largar (*drag and drop*) componentes do servidor para uma área de *design* que irá correr no servidor. Basta fazer um duplo *click* num componente do servidor, no *designer* de componentes, para que o código para esse objecto seja aberto.

- O novo *designer XML*.

O *designer XML* proporciona ferramentas intuitivas para trabalhar com *XML* e ficheiros *XSD* (*XML Schema Definition*). Dentro do próprio *designer* existem três vistas: uma para criar e editar esquemas *XSD*, uma para a alteração estruturada dos ficheiros de dados *XML*, e uma outra para alterar o código fonte *XML*.

A visualização do esquema, proporciona uma área visual de desenho na qual os programadores podem construir e editar virtualmente o esquema do *XML*. Novos esquemas podem ser criados no *designer* adicionando novos elementos, tipos e atributos ao editor ou arrastando tabelas do *Server Explorer* para a área de desenho. A visão do esquema permite ainda ao programador, criar relações entre as tabelas e gerar *datasets ADO .Net*.

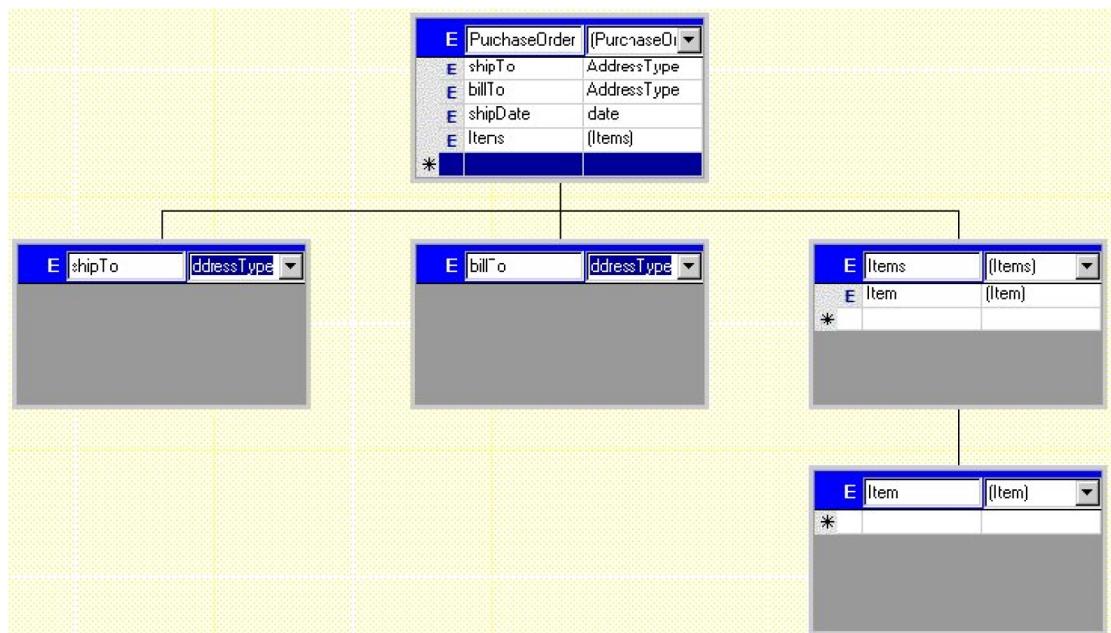


Figura 5.7 – O designer XML

A visualização dos dados está disponível quando um ficheiro de dados *XML* é adicionado a um projecto *Visual Studio .Net*. Utilizando a visualização dos dados, os programadores podem gerar, referenciar e visualizar esquemas associados a um ficheiro *XML*. Os programadores podem também visualizar e editar dados utilizando o visualizador de dados, o que torna fácil trabalhar directamente com dados baseados em *XML* tal como se estivessem numa base de dados.

- As novas macros *Visual Studio*.

O *Visual Studio .Net* vem equipado com um modelo de extensibilidade muito rico para customizar, automatizar e estender o ambiente de desenvolvimento integrado. Para aproveitar melhor este modelo de extensibilidade do *IDE*, o *Visual Studio .Net* proporciona um ambiente de macros *Visual Studio*.

Utilizando as macros *Visual Studio*, os programadores podem automatizar processos repetitivos para poupar tempo e esforço e para gravarem macros que mais tarde podem ser utilizadas para automatizar processos do *IDE*.

5.4 Construir a web programável

Com:

- *Forms web* e
- *Serviços web*.

Vamos agora então ver como se criam *forms web* e *serviços web*.

5.4.1 *Forms web*

Com as *forms web*, o *Visual Studio .Net* referencia a divisória existente entre as técnicas utilizadas para construir aplicações de *desktop* e aquelas utilizadas para criar aplicações *web*. Com o *Visual Studio .Net* e as *forms web*, os programadores podem rapidamente desenvolver aplicações *web* programáveis para diferentes plataformas e *browsers*, utilizando as mesmas técnicas previamente utilizadas para construir aplicações *desktop* baseadas em *forms*.

As tecnologias centrais para as *forms web* são providenciadas pelo *Framework .Net*, mas vamos agora fazer uma rápida revisão antes de continuarmos a falar de como o *Visual Studio .Net* torna a sua utilização bastante mais fácil.

Uma página *form web standard*, consiste de um ficheiro *markup form web* que contém, a representação visual, baseada em *XML*, da página e de um ficheiro fonte com código para tratamento dos eventos. A fonte é compilada para código executável. Ambos os ficheiros residem e são executados no servidor, onde geram uma página *web* que é enviada ao cliente. A página pode ser gerada como *HTML 3.2 puro*, o que quer dizer que pode ser visualizada em qualquer plataforma e em qualquer *browser*, ou então pode fazer referência às capacidades especiais de um *browser* específico. É inclusive possível referenciar dispositivos móveis que suportam *WML* e *WAP* utilizando *Adaptive UI Controls*.

No *Visual Studio .Net*, todas as linguagens dividem as mesmas tecnologias *web*. Independentemente de se escolher o *Visual Basic* ou o *C#*, a funcionalidade do produto é a mesma. Por isso, pode-se escolher a linguagem preferida para construir *forms web*.

5.4.1.1 A criação de uma *form web*

A programação de *forms web* é modelada pela maneira como os programadores que utilizam o *Visual Basic*, escrevem as suas aplicações tradicionais para o *Windows*. Por exemplo, hoje em dia construir uma *form Windows* com o *Visual Basic 6.0* requer a adição de uma *form* ao projecto, arrastar *controls* para a *form*, e depois fazer duplo *click* nos *controls* para escrever o código (por detrás da *form*). As *forms web* têm os mesmos princípios de desenho para a *web*.

Para se criar aplicações *web* utilizando *forms web*, adiciona-se uma página *form web* ao projecto, arrasta-se o *control* para a página e faz-se duplo *click* no *control* para se escrever o código por detrás da página. Os programadores podem escolher a linguagem que preferem para implementar o *bussiness logic* (“a lógica do negócio”) para a *form web*.



Figura 5.8 - Designer de form web distribuído

De uma perspectiva de desenho, as *forms web*:

- Utilizam um modelo de programação que é bastante familiar à grande maioria dos programadores. Os programadores podem movimentar-se facilmente entre projectos *desktop* e *web* sem ter a necessidade de formação extensiva.
- Separam o *layout HTML* do código por detrás da página. Esta separação torna mais fácil a actualização de ambas as peças independentemente uma da outra, e também simplifica a navegação no código.
- Podem ser criadas com qualquer linguagem que suporta o *Framework .Net* incluindo o *Visual Basic* e o *C#*.

5.4.2 Serviços *web*

Os serviços *web*, também já foram mencionados anteriormente.

Vamos então explicar como se cria, invoca e consome um serviço *web*.

O serviço que se vai criar é o de um serviço de cotação de mercado. Para simplificar, o serviço irá devolver *Buy* se o símbolo *ACME* for inserido.

5.4.2.1 Criar

- Começa-se por abrir o *Visual Studio* e criar um projecto serviço *web Visual Basic* ao qual se dá o nome “*Stocks*”.
- De seguida, abre-se o serviço *web* no modulo de visualização de código e escreve-se algum código (que tenha a ver com o serviço que pretendemos que ele execute).

3. Por último, escreve-se o atributo `<Webmethod()>` (já foi falado anteriormente) como sendo parte da definição da função.

E é tudo. Quando se carrega no botão *build* (compilação), o *Visual Studio* compila o código e envia-o para o servidor *web*. Adicionando o atributo `<Webmethod()>` à função, o compilador do *Visual Studio* dá instruções ao *Framework .Net*, para expor o método aos clientes *web*. Para além de invocar os métodos através do *URL* do serviço *web*, o *Framework .Net* expõe também duas características automaticamente. Se pedirmos ao *browser* o *URL*, o servidor irá retornar uma página de descrição *HTML* para o serviço *web*. Esta página mostra informação sobre os métodos disponíveis no serviço *web* e também permite que se invoquem métodos do *browser web*.

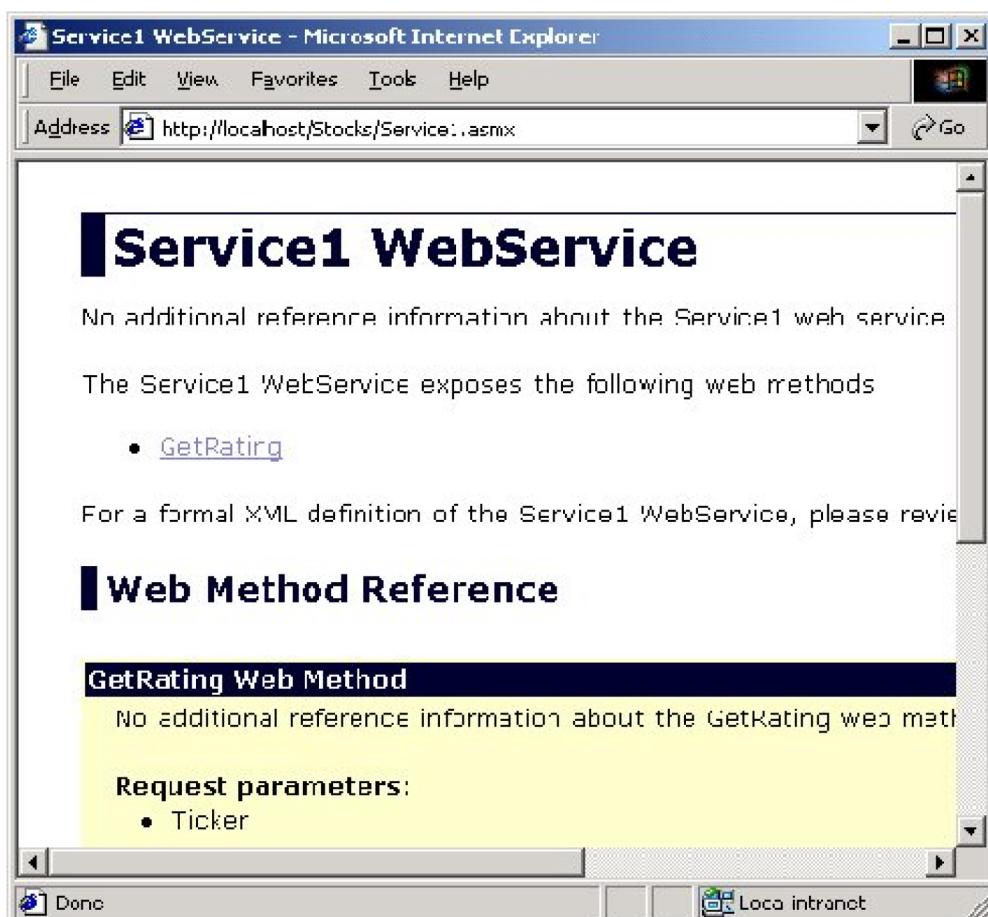
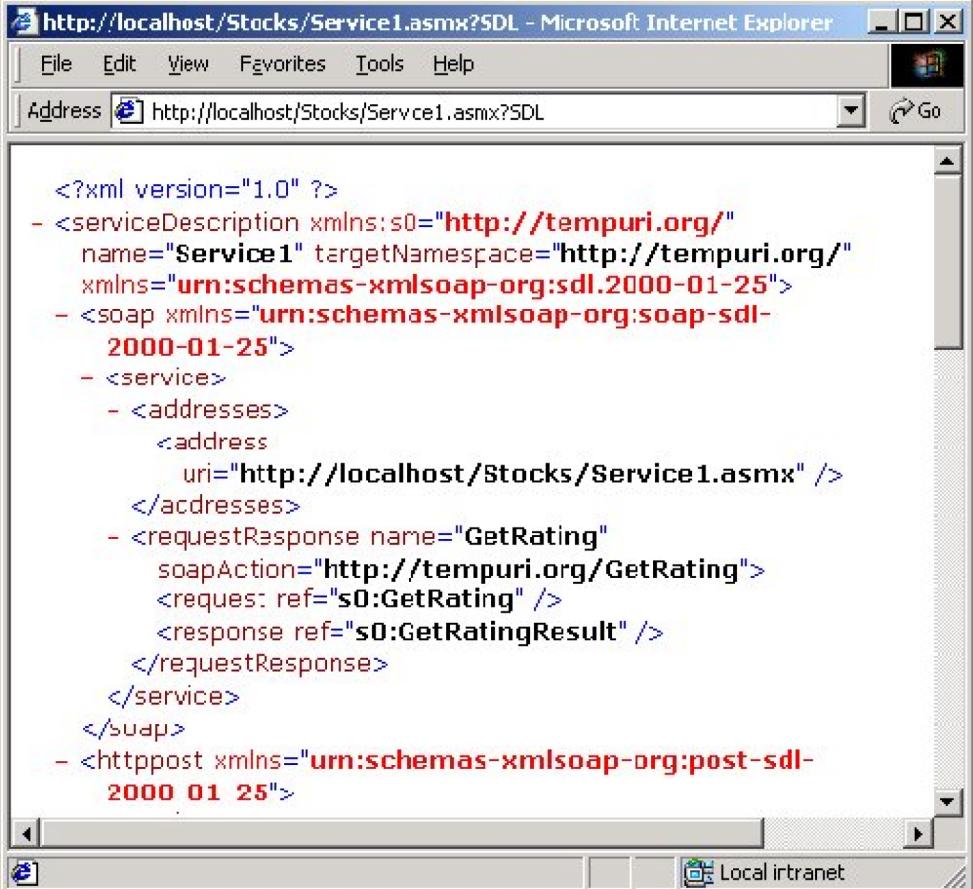


Figura 5.9 – Visualizando a página de descrição do serviço *web*

Se visualizarmos o serviço *web* com `?SDL` no final do *URL*, o servidor irá retornar um documento que descreve o serviço *web*. Este documento especifica as funções públicas do serviço, os parâmetros e tipos de dados, bem como os tipos de dados devolvidos. Isto é tudo criado automaticamente pelo *Visual Studio .Net*.



```

<?xml version="1.0" ?>
- <serviceDescription xmlns:s0="http://tempuri.org/" name="Service1" targetNamespace="http://tempuri.org/" xmlns="urn:schemas-xmlsoap-org:sdl.2000-01-25">
- <soap xmlns="urn:schemas-xmlsoap-org:soap-sdl-2000-01-25">
- <service>
- <addresses>
<address
    uri="http://localhost/Stocks/Service1.asmx" />
</addresses>
- <requestResponse name="GetRating"
    soapAction="http://tempuri.org/GetRating">
<request ref="s0:GetRating" />
<response ref="s0:GetRatingResult" />
</requestResponse>
</service>
</soap>
- <httpPost xmlns="urn:schemas-xmlsoap-org:post-sdl-2000-01-25">

```

Figura 5.10 – O documento *SDL* que é gerado automaticamente, especifica em *XML* como trabalhar com o serviço *web*.

5.4.2.2 Invocar

Depois do serviço *web* ter sido construído, pode ser invocado através de *HTTP* utilizando *XML* para passar os dados de e para o serviço. Pode-se testar uma função do serviço *web* directamente de um *browser*, passando o nome do método e os parâmetros através do *URL*. O *Framework .Net* irá empacotar automaticamente o resultado no *XML* que é retornado do serviço.



Figura 5.11 - Os serviços *web* retornam resultados *XML* para o *browser*

5.4.2.3 Consumir

Para utilizar um serviço web com o *Visual Studio*, só é necessário adicionar referencias web. O comando *Add Web Reference* lança uma caixa de dialogo que permite fazer isso. Depois de visualizar o URL do serviço web, é possível referenciar o serviço do projecto. Uma vez que o *SDL* do serviço web contém o URL do serviço bem como todas as funções que estão disponíveis, o *Visual Studio* pode criar automaticamente a “canalização” necessária para chamar o serviço a partir do código criado.

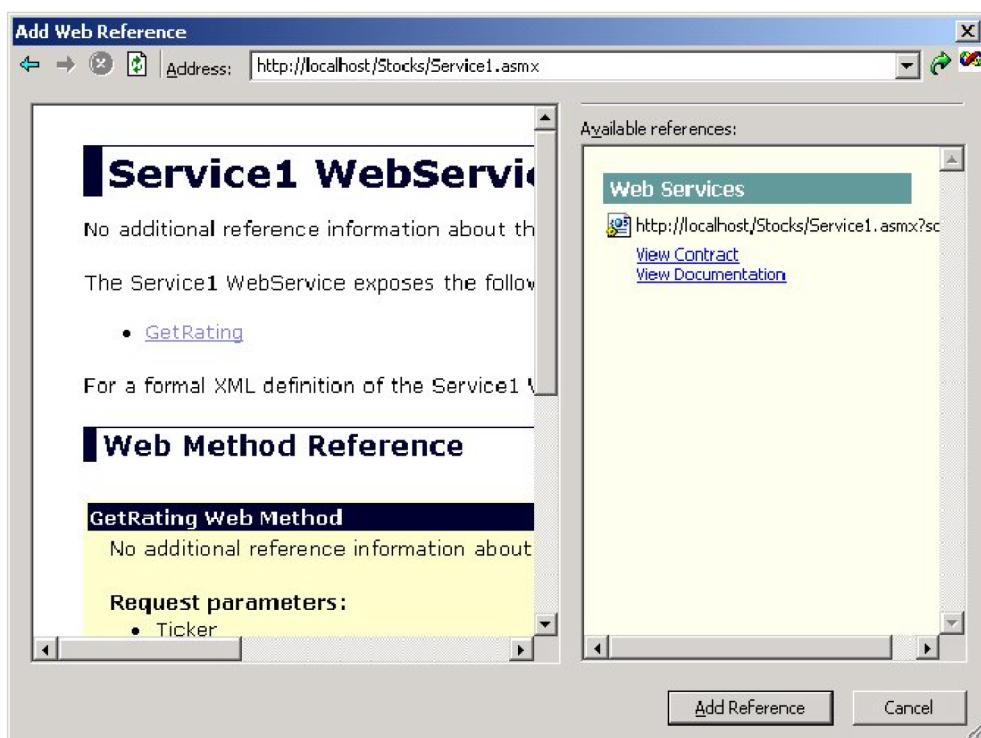


Figura 5.12 - Utilizando a caixa de dialogo do *Add Web Reference*

Uma vez que a referencia web esteja adicionada ao serviço, o *Visual Studio* providencia o “acabamento” de comandos, para aceder ás funções. Chamar um serviço web é tão simples como criar uma nova instancia da classe do serviço web e depois chamar os seus métodos.

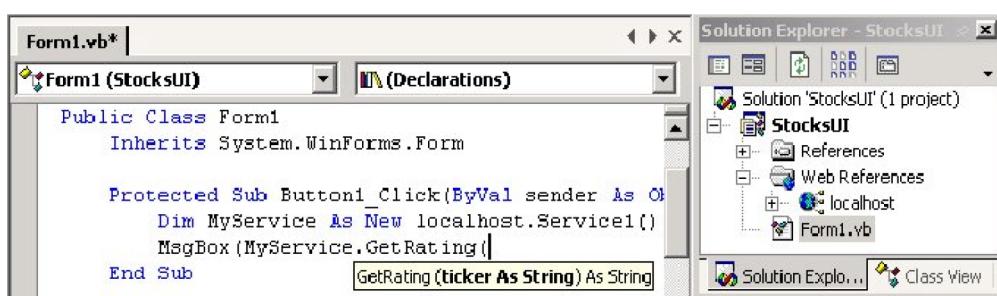


Figura 5.13 - O *Visual Studio* permite o acabamento de comandos nos serviços web

Devido ao facto de que os serviços *web* serem acessíveis utilizando *URLs*, *HTTP* e *XML*, os programas que correm em qualquer plataforma e em qualquer linguagem podem aceder aos serviços *web*. Os serviços *web* não precisam de ser escritos utilizando o *Visual Studio*, ou construídos utilizando o sistema operativo *Windows*. Por exemplo, o serviço de cotação de mercado que é mostrado acima, poderia ter sido criado utilizando o sistema operativo *UNIX* com um servidor *Web Apache*.

5.5 Visual Basic .Net

O *Visual Basic* foi sempre uma forma rápida e produtiva de criação de aplicações para o *Windows*. Nos últimos anos, os programadores que necessitam de desenvolver aplicações para a *web* rapidamente têm perguntado: “Onde está o *Visual Basic* para a *web*?”, e foi assim que apareceu o *Visual Basic .Net*.

Para além de ser muito melhor a criar soluções para clientes e servidores baseados no *Windows*, o *Visual Basic* é agora a maneira mais rápida e produtiva de criar aplicações *web* e serviços *web*.

5.5.1 Inovações na linguagem

As melhorias na linguagem do *Visual Basic .Net*, foram desenhadas para permitir construir, com o *Visual Basic*, aplicações *web* de confiança e escalonáveis.

5.5.1.1 Herança

A característica mais requisitada constantemente para o *Visual Basic*, é o suporte para herança. O desenvolvimento no tempo da Internet, requer uma rápida construção e reutilização massiva. O *Visual Basic* agora inclui uma implementação completa de herança.

Os programadores podem utilizar a nova palavra *Inherits* para derivar de uma classe existente.

```
Classe1
    Function BuscaCliente ()
    ...
End Function

Classe2
    Inherits Classe1
    Function BuscaOrdens()
    ...
End Function
```

Figura 5.14 – Exemplo de herança no *Visual Basic .Net*

O comando **Inherits** suporta todas as propriedades habituais associadas com a herança.

As instâncias da classe derivada suportam todos os métodos e interfaces que são suportadas pela classe base. E é claro, que a classe derivada pode estender o conjunto de métodos e interfaces suportados pela classe base.

A classe derivada pode sobre-escrever métodos definidos na classe base utilizando a palavra **Overrides**. Para poder reduzir o número de erros de programação, o *Visual Basic* não deixa que o programador accidentalmente sobre-escreva uma função; só funções que estão declaradas como *overrides* nas classes derivadas, têm permissão para sobre-escrever.

5.5.1.2 Overloading

O *Visual Basic* permite agora o *overloading* de funções, o que dá aos programadores a possibilidade de criar versões diferentes de uma função ou sub-função que têm o mesmo nome, mas têm tipos de argumentos diferentes.

O *overloading* é especialmente útil quando o modelo do objecto dita que se utilize nomes semelhantes para procedimentos que funcionam com tipos de dados diferentes. Por exemplo, uma classe que pode mostrar diversos tipos de dados diferentes, pode ter procedimentos **Display** que se parecem com isto:

```
Overloads Sub Display (theChar As Char)
...
Overloads Sub Display (theInteger As Integer)
...
Overloads Sub Display (theDouble As Double)
```

Figura 5.15 - Overloading

Sem o *overloading*, teria de se criar nomes distintos para cada procedimento ou utilizar um parâmetro **Variant**. O *overloading*, proporciona uma maneira mais explícita e eficiente de tratar tipos de dados múltiplos.

5.5.1.3 Construtores parametrizáveis

Construtores parametrizáveis permitem criar uma nova instância de uma classe e em simultâneo passar argumentos para a nova instância. Os construtores são essenciais para a programação orientada ao objecto uma vez que permitem que o código de construção definido pelo utilizador seja parâmetros passados pelo criador da instância. Eles simplificam o código do cliente, permitindo que uma instância de um novo objecto seja criada e inicializada numa única linha.

5.5.1.4 Tratamento de exceções estruturada

Desenvolver aplicações empresariais, requer a construção de componentes reutilizáveis e que tenham manutenção. Um aspecto da linguagem *Basic* que desafiou versões antigas, era o seu suporte de tratamento de erros. Os programadores tem vindo a reparar que um esquema consistente de tratamento de erros significa uma grande

quantidade de código duplicado. O tratamento de erros utilizando o comando existente **On Error GoTo**, por vezes atrasa o desenvolvimento e manutenção de grandes aplicações. O próprio nome reflecte alguns desses problemas: conforme é implícito no **GoTo**, quando ocorre um erro, o controle é transferido para uma localização que tem o **label** dentro da subrotina. Uma vez que o código do erro seja executado muitas vezes tem de ser desviado, via uma outra localização de limpeza, via o standard **GoTo**, que por fim utiliza um outro **GoTo** ou um **Exit** para sair do procedimento. Tratar de diversos erros diferentes com várias combinações de **Resume** e **Next** produz rapidamente código ilegível e levam a *bugs* frequentes quando os caminhos de execução não foram completamente pensados do princípio ao fim.

Com o **Try...Catch...Finally**, este problema desaparece e os programadores podem juntar o seu tratamento de exceções, e existe uma estrutura de controle para escrever código limpo que executa tanto em condições normais, como em exceções.

```
Sub SEH()
    Try
        Open "TESTFILE" For Output As #1
        Write #1, CustomerInformation
    Catch
        Kill "TESTFILE"
    Finally
        Close #1
    End try
End Sub
```

Figura 5.16 – Um exemplo do **Try...Catch... Finally**

5.5.2 Características adicionais da linguagem modernizada

A Microsoft adicionou um número de interpretações adicionais que simplificam a programação de aplicações mais robustas e escalonáveis. Algumas dessas características são:

- **5.5.2.1 Verificação severa de tipos.**

A linguagem do *Visual Basic* de hoje, é muito liberal na coerção dos tipos implícitos que vai gerar. Para a transferencia e passagem de parâmetros sem ser por referencia, o compilador do *Visual Basic* permite praticamente que qualquer tipo de dado seja convertido para qualquer outro tipo de dado, com a geração de coerções *runtime*. A operação de coerção *runtime*, falha se o valor que vai ser transferido não o possa ser sem perda de informação. Com a adição de uma nova opção de compilação, o *Visual Basic* pode gerar erros em tempo de execução, para qualquer conversão que possa dar erro em *runtime*. O **Option Strict** melhora a segurança dos tipos (de dados) gerando erros quando é necessária uma conversão que pode falhar em *runtime* ou que seja inesperado pelo utilizador, como a conversão automática entre tipos numéricos e *strings*.

- **5.5.2.2** Membros distribuídos.

Os membros distribuídos são membros de classes, de dados e funções, que são divididos por todas as instâncias da classe. Dividir uma única instância de um membro de dados ou função, por todas as instâncias das classes, é necessário num *Visual Basic* com herança. Um membro de dados distribuído, é um método, que ao contrário de métodos normais, não é implicitamente passado como instância de uma classe. Por esta razão, não é permitido que nenhuma referência não qualificada, para membros de dados não distribuídos, seja um método distribuído.

- **5.5.2.3** Inicializadores.

O *Visual Basic* suporta inicialização de variáveis na linha em que são declaradas. Os inicializadores podem ser utilizados em qualquer lado incluindo dentro de uma estrutura de controle. A semântica de uma declaração ao nível do procedimento, que inclui um inicializador, é a mesma que o comando de declaração que é imediatamente seguido de um comando de atribuição. Ou seja, isto:

```
Dim X As Integer = 1
```

Figura 5.17 – Inicializador

É igual a isto:

```
Dim X As Integer  
X = 1
```

Figura 5.18 – Código normalmente utilizado no *Visual Basic*

5.5.3 A ferramenta de *upgrade*

O *Visual Basic .Net* inclui um *wizard* para ajudar os programadores a fazerem o *upgrade* dos seus projectos *Visual Basic 6.0*. O *wizard* é invocado quando o programador abre pela primeira vez um projecto *Visual Basic 6.0* no *Visual Basic .Net*. O *wizard* apresenta ao programador opções para controlar o processo de *upgrade*.

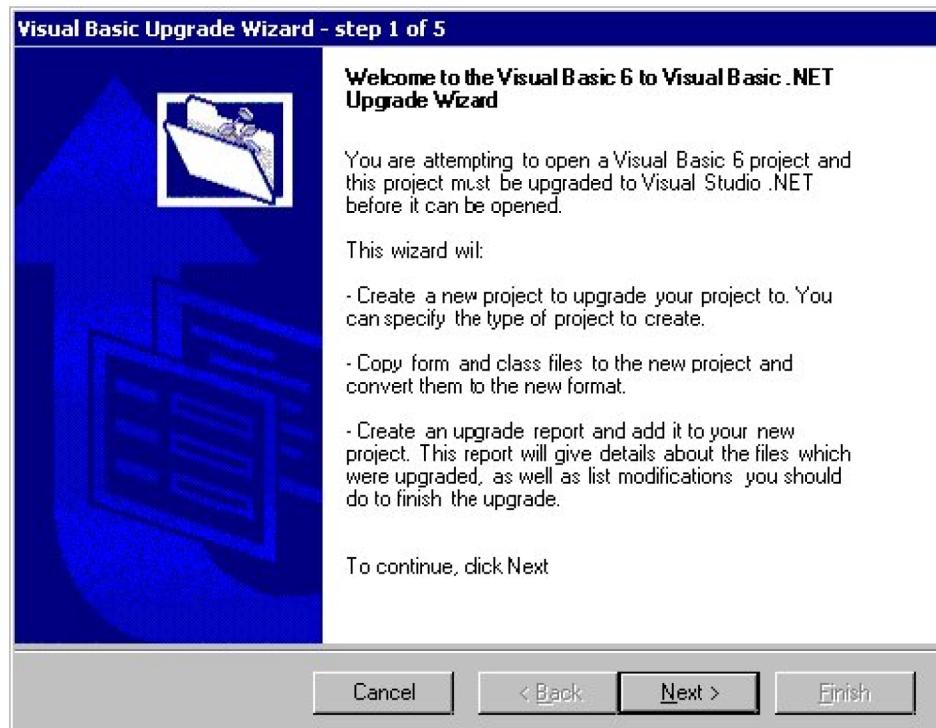


Figura 5.19 – A ferramenta de *upgrade*

Quando é executado, o *wizard* cria uma nova versão *Visual Basic .Net* do projecto, deixando a versão *Visual Basic 6.0* original, exactamente como estava.

Devido às diferenças de linguagem e do pacote das forms entre as versões do *Visual Basic*, a maioria dos projectos não podem ser *upgraded* automaticamente, com uma exactidão de 100%. O *wizard* de *upgrade* ajuda a identificar as áreas que requerem atenção, providenciando um relatório detalhado, que mostra qualquer código que necessite de ser arranjado manualmente.

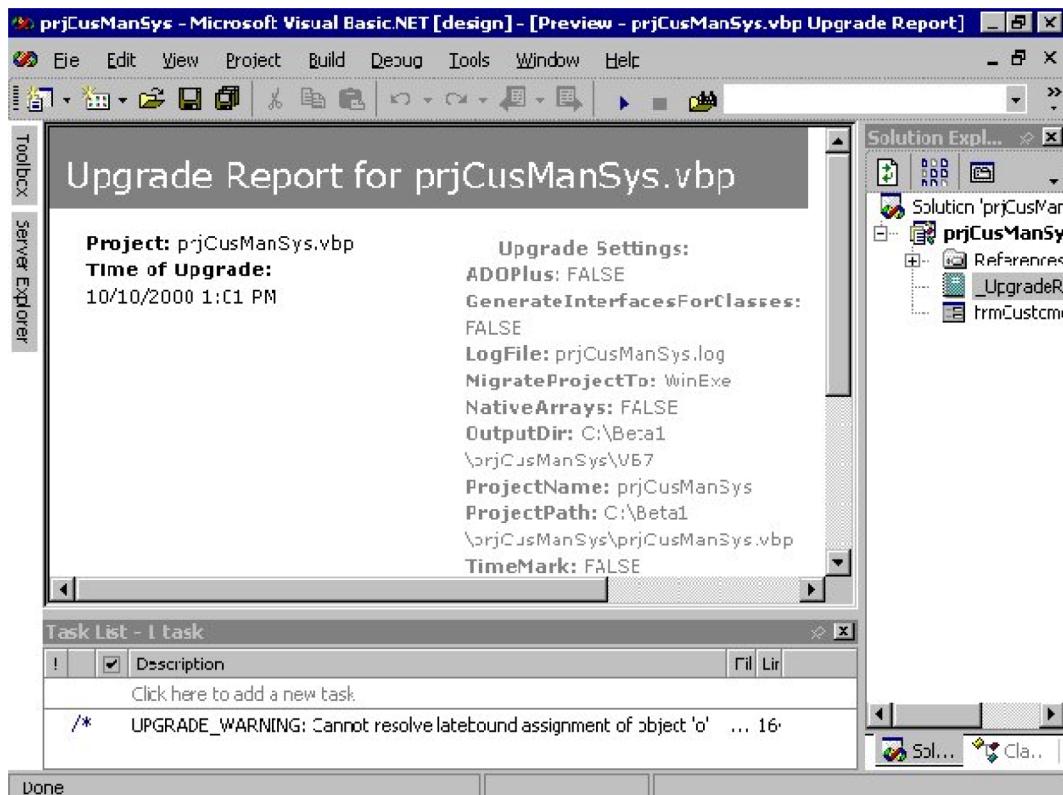


Figura 5.20 – Relatório do *upgrade*

Qualquer mudança necessária (isto é, código que poderá causar erros de compilação) aparece na lista de tarefas, dando ao programador uma *checklist* com a qual pode trabalhar. O relatório também inclui avisos (*warnings*) que apontam qualquer código que, embora não impeça o código de ser compilado, possa causar outros potenciais erros. Por fim, são adicionadas sugestões ao relatório, onde o código possa ser melhorado, para tirar partido das novas capacidades do *Framework .Net*.

Para cada erro, aviso (*warning*), ou sugestão no relatório é providenciado um *link* para um tópico de ajuda que oferece uma explicação detalhada do problema, bem como delineia os passos necessários para o resolver. Conforme o programador trabalha pelos itens do relatório, ele também aprende sobre muitas das diferenças entre as duas versões do *Visual Basic*.

5.5.4 *Upgrade* de conhecimentos

Como com qualquer *upgrade*, os programadores devem-se preparar para investir algum tempo para aprenderem os novos conceitos e características do *Visual Basic .Net*. Ajuda extensiva está disponível na forma de documentação que detalha todas as mudanças entre as duas versões, e que está especificamente apontada para os programadores que estão a fazer o *upgrade*. Para cada mudança no produto, um tópico de ajuda é disponibilizado, explicando “se você fizer X no *Visual Basic 6.0*, aqui está como faz no *Visual Basic .Net*”.

Por exemplo, os programadores que utilizam o *Visual Basic 6.0*, utilizam frequentemente *control arrays* para tratar de eventos relacionados para um grupo de *controls*. Os *control arrays* não são suportados no *Visual Basic .Net* devido á

arquitectura do *Framework .Net*; existe um tópico de ajuda que mostra como conseguir o mesmo, utilizando eventos delegados.

Uma das inovações mais importantes no *.Net*, é a maneira como unifica os modelos de programação para diferentes tipos de trabalho, em particular, programar para aplicações de Internet é praticamente igual à programação de aplicações locais baseadas em *forms*. Aqui temos um exemplo de como os dois são semelhantes.

Vamos criar um programa para calcular quantos dias passaram desde o aniversário de uma pessoa.

Abre-se o *Visual Basic .Net* e escolhe-se a opção **Windows Application**. No **Form1** que aparece, adiciona-se os seguintes *controls* e defini-se as propriedades conforme a tabela:

Control	Propriedades
Label	Text = "Birthday"
TextBox	Text = "" Name = "txtBirthday"
Button	Text = "Calculate days"
Label	Name = "lblDaysAlive" Text = ""

Tabela 5.1

Agora faz-se duplo *click* sobre o botão, e coloca-se o código que se segue, na rotina do evento *click*.

```
Dim datBirthDate As Date  
DatBirthDate = Cdate(txtBirthday.Text)  
Dim nDaysAlive As Integer  
NDaysAlive = DateDiff(DateInterval.Day, datBirthDate, - System.DateTime.Today)  
LblDaysAlive.Text = CStr(nDaysAlive)
```

Figura 5.21 – Código a ser colocado na rotina do evento *click* do botão

Faz-se *run* ao programa, e coloca-se uma data válida na *textbox*. Pressiona-se o botão e o resultado do ecrã irá ser qualquer coisa como:

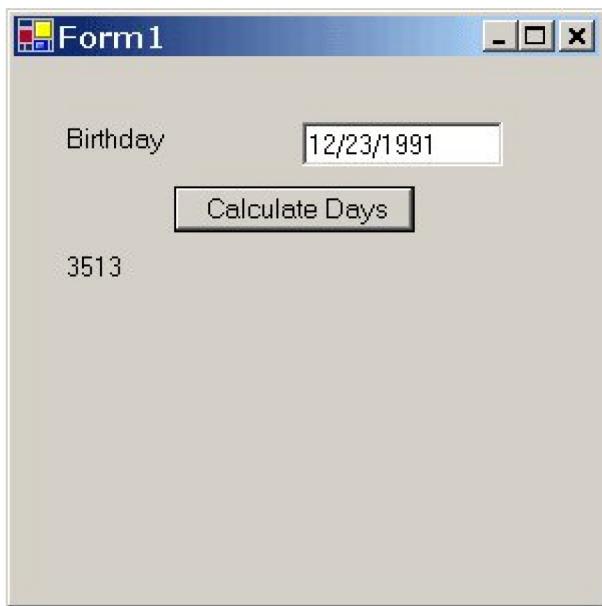


Figura 5.22 – Ecrã visualizado ao fazer-se *run* do programa

Como dá para ver, é exactamente igual a como seria feito no *Visual Basic 6.0*. Mas é aqui que as coisas se tornam mais interessantes.

Abre-se um novo projecto, e escolhe-se a opção **ASP .Net Web Application**. Na **WebForm1** que aparece, seguem-se exactamente os mesmos passos que acima. Ou seja, colocam-se os mesmos quatro *controls* na *form* e o mesmo código no evento *click* do botão.

Agora obtém-se um ecrã como o abaixo, com a interface a ser apresentada no *Internet Explorer*.

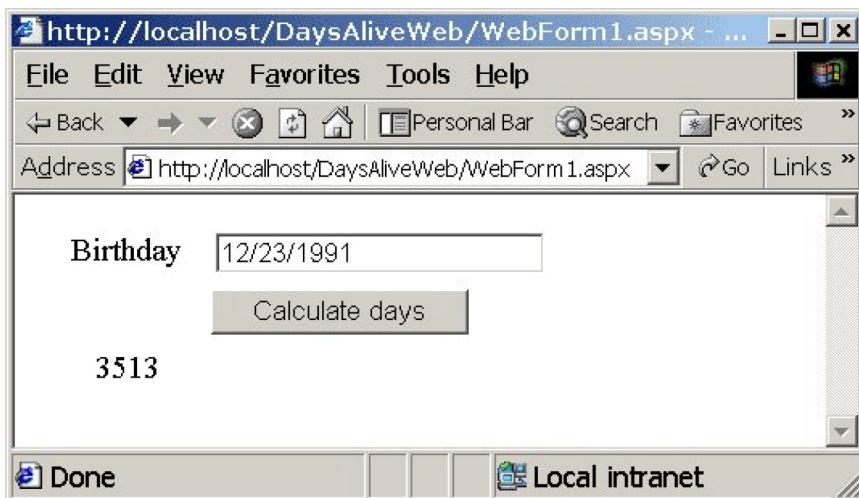


Figura 5.23 – Form web apresentada no *Internet Explorer*

Existem umas diferenças mínimas no desenvolvimento destes dois programas semelhantes. Por exemplo, o botão na *form web* faz o *resize* automático quando se coloca texto que é demasiado grande para ser visualizado. Mais, o texto por defeito na *textbox* é uma *string* nula, para as *forms web*, enquanto nas *forms Windows* é o nome do *control*. Contudo isto são apenas pequenas diferenças. Em geral o modelo

conceptual para o programa de desenvolvimento é o mesmo para ambas as *forms Windows* e *forms web*.

5.6 Visual C++ .Net

O *Visual C++* contém uma riqueza de características novas e melhoradas que permitem aos programadores criar e consumir aplicações *web* e serviços *web*. O *Visual C++* esforça-se bastante para ajudar os programadores a produzir programas mais pequenos, rápidos e mais funcionais, em menos tempo.

Os programadores escolhem tipicamente o *C++* como linguagem de desenvolvimento porque ela proporciona-lhes mais poder e flexibilidade do que qualquer outro ambiente de desenvolvimento. Os programadores de *C++* querem o máximo controle sobre os componentes que fazem parte da sua aplicação, e esperam que as suas ferramentas de desenvolvimento produzam os programas mais compactos e da melhor *performance* possível. O *Visual C++* torna mais fácil aos programadores atingirem esses objectivos.

5.6.1 Poder e flexibilidade

O *Visual C++* é único entre as linguagens *.Net*, uma vez que suporta o modelo de código manejável, providenciado pelo *Framework .Net*, bem como um modelo de código não manejável. Suportando ambos os modelos de programação, o *Visual C++* preserva e melhora os investimentos existentes e proporciona o máximo de escolhas possíveis para os programadores.

O *Visual C++* inclui uma muito poderosa extensão ao *Active Server Library (ATL)* chamado servidor *ATL*, que ajuda os programadores a criar aplicações e serviços *web* compactos e de alta *performance*. O servidor *ATL* aproveita os melhores hábitos de desenvolver aplicações *web* de alta *performance*, e condensa esses hábitos num conjunto de classes *ATL* simples e extensíveis para os programadores reutilizarem.

Os melhoramentos ao *Visual C++* tornam fácil aos programas do lado do cliente ou servidor-a-servidor, invocar outros serviços *web*, quer se esteja a utilizar o *Framework .Net*, *MFC*, *ATL* ou a programar directamente para os *APIs* do *Windows*.

O *Visual C++* também inclui um numero de características que tornam mais fácil manter e melhorar as aplicações baseadas no *Windows* existentes. Actualizações ao *MFC* dão possibilidade aos programadores, de aceder aos melhoramentos mais recentes da plataforma *Windows* e existem também novas optimizações ao compilador, que produz código mais pequeno e rápido do que nos lançamentos anteriores.

5.6.2 Servidor *ATL*

Quando os programadores escolhem escrever parte das suas aplicações *web* em *C++*, eles normalmente fazem-no por razões de *performance* e controle. Soluções *web* escritas em *ASP .Net* funcionam muito bem e serão mais do que potentes para a maioria das aplicações *web*. No entanto, há vezes em que o programador irá necessitar daquele bocadinho mais de *performance*, que o código em *C++* proporciona, ou o

programador pode querer controle completo sobre todas as partes do código de um componente critico. O servidor *ATL* foi precisamente desenhado para estas situações.

O servidor *ATL* aproveita os melhores hábitos para desenvolver as melhores aplicações *web* de alta performance e condensa esses hábitos num conjunto de classes *ATL* simples e extensível para os programadores reutilizarem.

5.6.2.1 Como funciona o servidor *ATL*

O servidor *ATL* encoraja a criação de aplicações *web* de uma maneira modular. Uma aplicação pode ser uma combinação de ficheiros *HTML* simples e de ficheiros *stencil* *HTML* especialmente marcados, que têm uma extensão *SRF* (*Server Response File*). O *HTML* no ficheiro *stencil* é algo parecido a:

```
<body>
<p>{(HelloWorld)}</p>
</body>
```

Figura 5.24 – HTML no ficheiro *stencil*

As chavetas duplas a volta do *HelloWorld* marcam-no como uma *tag* que necessita de processamento especial. O servidor *ATL* processa o ficheiro *stencil* e converte as *tags* em chamadas aos métodos associados as páginas *HTML*. Quando o método retorna a string “Hello World!” então a página *HTML* apresenta o seguinte:

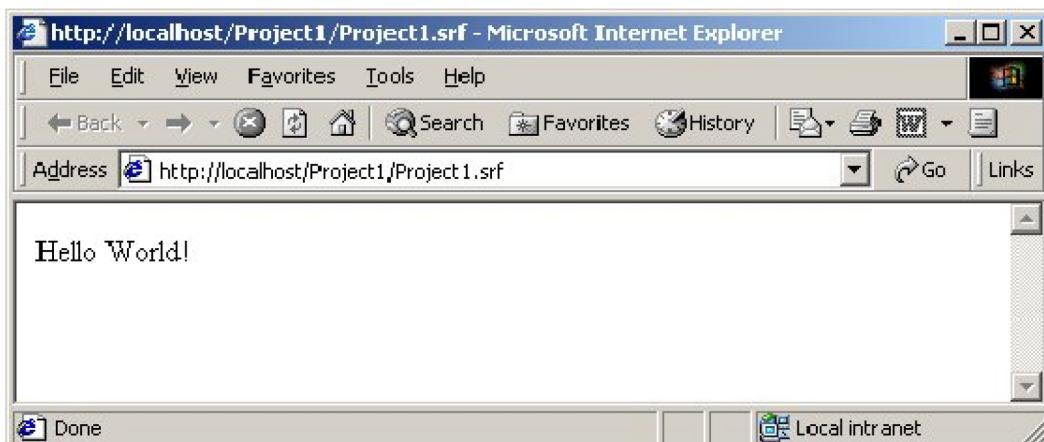


Figura 5.25 – Uma aplicação simples do servidor *ATL*

Para além de ficheiros *HTML* e *SRF* simples, também se pode misturar páginas do servidor *ATL* com *ASP* e páginas *ASP .Net* na mesma aplicação. Isto permite aos programadores, misturar e encontrar tecnologias de implementação baseadas nas suas necessidades.

Embora os *frameworks* sejam diferentes, os serviços *web* escritos em *ASP .Net* e servidor *ATL*, comportam-se exactamente da mesma maneira em respeito ao cliente. Por causa do modelo consistente, os programadores podem utilizar o *framework* que melhor se enquadra para o seu projecto, sem que os programas clientes, sejam afectados.

5.6.2.2 Performance e escalonabilidade

O servidor *ATL* simplifica em muito o processo de escrever aplicações web bastante escalonáveis e de elevada performance. As aplicações do servidor *ATL* correm dentro do *IIS* como *DLLs ISAPI*. A *DLL* para uma aplicação é carregada uma única vez na sua primeira invocação, e todas as invocações sucessivas são chamadas muito simples e eficientes a funções.

O servidor *ATL* também tem um suporte interno para o *thread pooling*, que também aumenta a performance e aumenta a disponibilidade do servidor. Devido ao facto do *IIS* ter um número limitado de *threads* para tratamento de pedidos, o servidor *ATL* criou a sua própria *pool* de *threads* para utilizar quando está a gerar a página que irá ser devolvida ao cliente. O pedido do *IIS* é manuseado separadamente para este *thread pool*, no qual o objecto do programador é invocado para criar a página. O resultado é então devolvido ao *IIS*, para ser devolvido ao cliente.

Esta arquitectura permite que as aplicações web e serviços web escritos com o servidor *ATL* aumentem muito e para além de tudo são transparentes para o utilizador. Devido ao facto de todas as aplicações de servidor terem requisitos de performance diferentes, uma vantagem chave para os programadores é o facto de o *thread pooling* e de os outros componentes do servidor *ATL*, poderem ser afinados precisamente para as necessidades da aplicação.

O servidor *ATL* também oferece muitos outros serviços adicionais. Um exemplo poderá ser a característica de guardar estados de sessão entre pedidos de cliente.

5.6.2.3 Um melhor suporte do servidor de aplicações

O servidor *ATL* tem um certo numero de outras características que foram especificamente desenhadas com o servidor de aplicações em mente. Existe um pequeno e muito eficiente cliente *HTTP* desenhado para aplicações servidor-paraservidor. O cliente *HTTP* permite aos programadores invocar outros serviços web do seu código para o servidor, sem o *overhead* de implementações *HTTP* maiores e focadas para o cliente.

O servidor *ATL* também inclui classes que ajudam a gerar construções simples de *HTML* a partir de código, e inclui também um simples serviço de correio *SMTP* que permite, por exemplo, mensagens de estatutos do servidor de *mail* para outra máquina, ou o envio de uma confirmação de um pedido para um cliente.

O servidor *ATL* inclui contadores de performance internos que permitem fazer estatísticas básicas, como contagens de chamadas de páginas, sem ter de escrever qualquer tipo de código adicional. Também estão incluídas classes que tornam muito fácil a criação de contadores de performance que se ligam aos serviços de monitorização do *Windows 2000*.

5.6.2.4 Totalmente configurável

Uma vez que está incluído o código fonte completo, os programadores estão completamente livres de configurar e modificar por completo o servidor *ATL*, dependendo das suas necessidades.

Para além disso, o servidor *ATL* foi desenhado de uma maneira modular para que muitos dos componentes chave, tais como manuseamento de *thread* e do estado da sessão, sejam particularmente fáceis de customizar. Os programadores podem trocar

unicamente essas peças individuais, e continuar a utilizar o resto do *framework* sem modificações.

5.6.3 Extensões manejáveis para o C++

As extensões manejáveis para o C++ são um conjunto de extensões de linguagens que ajudam os programadores, que utilizam o *Visual C++*, a escrever aplicações para o *Framework .Net*.

As extensões manejáveis são úteis se se quiser:

- melhorar a produtividade do programador, utilizando para escrever aplicações *Framework .Net*, o C++;
- migrar em partes uma grande porção de código, de C++ não manejável para o *Framework .Net*;
- ter componentes C++ não manejáveis que se quer utilizar de aplicações *Framework .Net*;
- ter componentes *Framework .Net*, que se quer utilizar a partir de C++ não manejável;
- querer misturar código C++ não manejável e código *Framework .Net* na mesma aplicação.

As extensões manejáveis para o C++ oferecem flexibilidade sem paralelo aos programadores que têm como objectivo *Framework .Net*. O código C++ não manejável e manejável tradicional, pode ser livremente misturado na mesma aplicação. Novas aplicações escritas com extensões manejáveis, podem tirar partido do melhor de ambos os mundos. Os componentes existentes, podem ser facilmente sobrepostos como componente *Framework .Net* utilizando as extensões manejáveis, preservando assim os investimentos no código existente, enquanto se integram com o *Framework .Net*.

5.6.3.1 O que são extensões manejáveis?

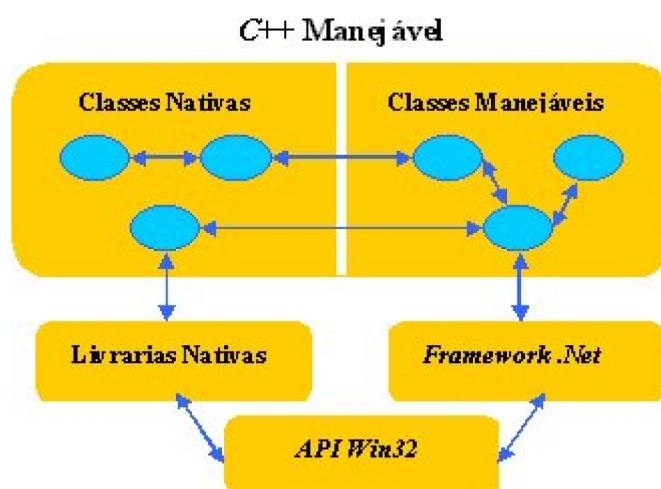


Figura 5.26 - C++ Manejável

As extensões permitem que se escreva classes manejáveis em C++, que correm no *Framework .Net*. (Referencias a “classes C++ não manejáveis” ou “C++ não

manejável” referem-se a código tradicional *C++*, que corre directamente no ambiente baseado em *Windows* tradicional, e não no *Framework .Net*.) As classes manejáveis conseguem tirar o máximo partido do *Framework .Net*.

A extensões manejáveis são palavras chaves e atributos novos no sistema de desenvolvimento *Visual C++*. Eles permitem que o programador decida que classes e que funções se compilam como código manejável ou não manejável. Estas peças, podem inter-operar, suavemente, umas com as outras e com livrarias externas.

As extensões manejáveis também são utilizadas para expressar tipos e conceitos do *Framework .Net*, directamente em código fonte *C++*.

5.6.3.2 Uma migração suave do código existente para o *Framework .Net*

Se se tem um grande investimento em código *C++*, as extensões manejáveis, irão ajudar a fazer uma transição suave para o *Framework .Net*. Devido ao facto de se poder misturar código manejável e não manejável na mesma aplicação – e até no mesmo ficheiro – pode-se em tempo mudar o código, componente a componente, para o *Framework .Net*. Ou pode-se continuar a escrever componentes com o *C++* não manejável, tirando assim partido completo do poder e flexibilidade da linguagem, e utilizar as extensões manejáveis só para escrever adaptadores de alta performance que tornam o código *C++* “chamável”, de componentes *Framework .Net*.

5.6.3.3 Aceder a um componente *C++* de uma linguagem *Framework .Net*

As extensões manejáveis permitem que se chame uma classe *C++* de qualquer linguagem *Framework .Net*. Só se necessita de escrever uma simples classe “empacotadora”, utilizando as extensões que expõem as classes e métodos *C++* como classes manejáveis. A “empacotadora”, é uma classe completamente manejável e pode ser chamada de qualquer linguagem *Framework .Net*. A classe “empacotadora” actua como um nível de mapeamento entre a classe manejável e a classe *C++* não manejável. Passa simplesmente chamadas a métodos do mundo *Framework .Net*, directamente para a classe não manejável. As extensões não manejáveis podem ser utilizadas para chamar qualquer livraria *link-dinâmica (DLL)* nativa, bem como as classes nativas.

5.6.3.4 Aceder a classes *Framework .Net* a partir de código nativo

Utilizando as extensões manejáveis, pode-se criar e chamar uma classe *Framework .Net* directamente de código *C++*. Pode-se escrever código *C++* que trata o componente *Framework .Net* tal e qual qualquer outra classe *C++* manejável. Também pode ser utilizado o suporte nativo *Component Object Model (COM)* no *Framework .Net* para chamar classes *Framework .Net*. Quer se utilize *COM* ou extensões manejáveis, para aceder a componentes *Framework .Net*, irá depender unicamente do projecto.

5.6.4 Programação de atributos

A programação de atributos é um dos melhoramentos da linguagem *C++* que permite aos programadores focarem na lógica do seu código, em vez de nos detalhes de implementação. Em alguns casos, tais como aqueles com *IDL*, elimina a necessidade de ficheiros externos de descrição ajudando assim, nas preocupações sobre a manutenção. Como resultado, os atributos servem para aumentar a produtividade dos

programadores C++. Alguns dos benefícios proporcionados por esta nova tecnologia, incluem:

- uma convenção de chamadas familiar que é conforme a estrutura clássica da linguagem C++;
- derivação fácil de classes base sem detalhes de implementação caóticos;
- substituir grandes quantidades de código fonte, com poucas e simples declarações.

Os atributos simplificam o processo de desenvolvimento gerando código automaticamente para o programador. Tal como é com o *ATL* ou *MFC*, o código gerado está sempre disponível aos programadores, caso seja necessário perceber exactamente que operações o código está a executar.

5.6.4.1 Tipos de atributos

Devido ao facto de os atributos serem intencionados a encapsular conceitos de domínio de uma maneira declarativa, existe um certo número de categorias de atributos:

- Atributos de servidor *ATL*
Assistem na escrita de aplicações web de alta performance com o servidor *ATL*. Por exemplo, os atributos são utilizados para marcar métodos, que podem ser chamados como serviços web.
- Atributos *COM*
Suportam inúmeras áreas do desenvolvimento *COM*.
- Atributos *IDL*
Os atributos *IDL* criam ou modificam o ficheiro *IDL* associado por dentro de um ficheiro de código fonte sem um *wizard* e sem ser familiar com a estrutura e sintaxe desse mesmo ficheiro. Colocando atributos *IDL* no ficheiro fonte C++, os programadores podem manter a informação numa única localização, para uma manutenção mais fácil.
- Atributos de acesso aos dados
Os atributos de acesso aos dados irão automaticamente adicionar código, baseado nos *OLE DB Consumer Templates*, que cria uma classe cliente *OLE DB*. O código auto-gerado efectua tarefas tais como abrir tabelas, executar comandos e aceder a dados.
- Atributos de eventos
Encapsula diferentes modelos de eventos numa interface comum, para tornar mais fácil aos programadores, manusearem diversos eventos, de tipos diferentes, numa maneira standard.

5.6.4.2 Utilizar os atributos

Podem-se inserir atributos num projecto, colocando-os manualmente no código fonte ou utilizando os *wizards* do *Visual C++*, que irão inseri-los eles próprios.

Na altura da compilação, o compilador irá reconhecer a presença de atributos no ficheiro fonte e conseguirá compila-los e verifica-los dinamicamente. Quando o projecto é compilado, o compilador compila cada ficheiro fonte C++, produzindo um ficheiro objecto. No entanto, quando o compilador encontra um atributo, ele é compilado e verificado sintaticamente. O compilador então, chama dinamicamente um *provider* de atributos para inserir código ou fazer outras alterações em tempo de execução.

A figura a seguir, demonstra a relação entre o compilador e o *provider* de atributos.

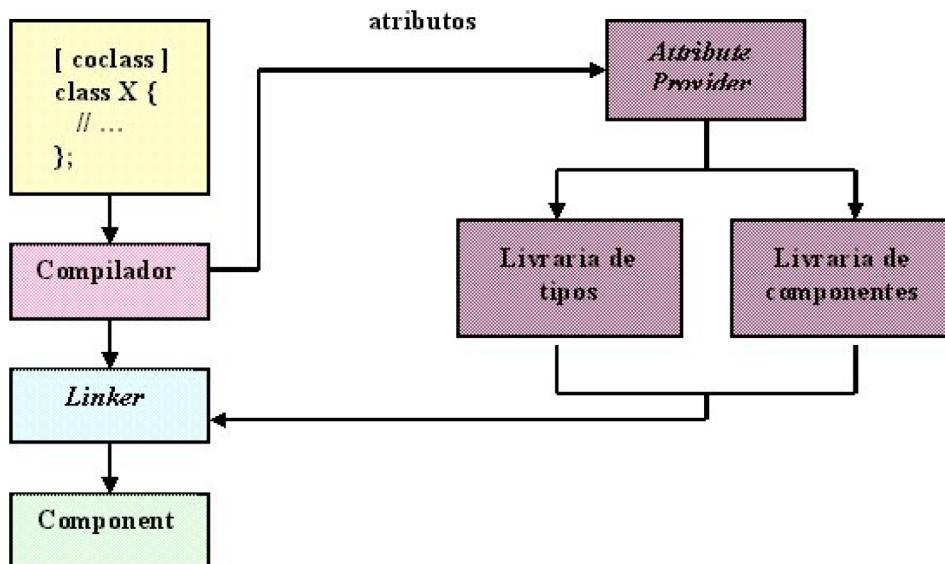


Figura 5.27 – Relação entre o compilador e o *provider* de atributos

A utilização de atributos não altera o conteúdo do ficheiro fonte. O programador pode gerar um ficheiro separado para cada ficheiro fonte no projecto, que mostra os resultados da substituição do atributo. Isto permite que se examine o código que o compilador está a gerar.

Os atributos são utilizados extensivamente nas novas características do *Visual C++*, tais como o servidor *ATL*. Eles permitem aos programadores escrever menos código, e também permitem que se desenvolva de uma maneira mais declarativa. Os atributos simplificam o processo de desenvolvimento permitindo aos programadores focarem na lógica do seu código em vez dos detalhes de programação.

5.6.5 Escrever código robusto

O *Visual C++* inclui uma característica nova para detectar quando um programador inadvertidamente escreve dados para além do *buffer*, isto é um erro conhecido por *buffer overrun*. Estes são alguns dos erros mais comuns que causam o *crash* do código, é também uma das maneiras principais utilizadas para atacar servidores web na Internet.

O *Visual C++* pode ser configurado para verificar a integridade dos dados escritos para um *buffer* alocado dinamicamente.

5.6.6 Performance pura

O compilador do *Visual C++*, um dos melhores do mundo a gerar código pequeno (pouco de quantidade) e rápido, inclui uma melhoria chamada *Whole Program Optimization* ou seja “optimização do programa inteiro”. Uma vez que o compilador C++ compila os ficheiros fonte, um de cada vez, ele só consegue optimizar dentro de um único ficheiro. O compilador e *linker* agora comunicam para permitir optimização

a ficheiros fonte múltiplos. Isto permite que o compilador faça melhoramentos a código pelo programa inteiro, o que não era possível anteriormente.

5.7 C#

Nas passadas duas décadas, o *C* e o *C++* têm sido as duas línguas mais utilizadas para desenvolver *software* comercial e para negócios. Ambas as linguagens proporcionam ao programador uma quantidade tremenda de controlo muito refinado, contudo esta flexibilidade tem os seus custos: a produtividade. Comparada a uma linguagem como o *Visual Basic*, as aplicações equivalentes em *C* e *C++* levam mais tempo a desenvolver. Devido à complexidade e tempos de ciclos grandes associados a estas linguagens, muitos programadores de *C* e *C++*, têm procurado uma linguagem que oferece um equilíbrio melhor entre poder e produtividade.

Existem linguagens hoje, que aumentam a produtividade sacrificando a flexibilidade de que os programadores de *C* e *C++* tanto necessitam. Tais soluções constrangem demasiado o programador. Elas não inter-operam facilmente com os sistemas preexistentes, e muitas vezes não se misturam muito bem com os hábitos de programação web correntes.

A solução ideal para os programadores de *C* e *C++* seria um desenvolvimento rápido combinado com o poder de aceder a todas as funcionalidades da plataforma existente por debaixo. Eles querem um ambiente que está completamente sintonizado com standards web emergentes e que proporcionam uma integração fácil com aplicações existentes.

A solução da *Microsoft* para este problema é uma linguagem chamada *C#* (é pronunciado *C sharp*). O *C#* é uma linguagem moderna e orientada ao objecto, que permite aos programadores construir rapidamente um grande leque de aplicações para o novo *Framework .Net*, que providencia ferramentas e serviços que exploram por completo a computação e comunicação.

Utilizando simples construções da linguagem *C#*, os componentes podem ser convertidos em serviços web, permitindo-lhes serem invocados através da Internet, a partir de qualquer linguagem, a correr em qualquer sistema operativo

Acima de tudo, o *C#* foi desenhado para possibilitar um desenvolvimento mais rápido ao programador de *C++* sem sacrificar o poder e controlo que têm sido a marca do *C* e do *C++*. Devido a esta herança, o *C#* tem um grau elevado de fidelidade com o *C* e o *C++*. Os programadores familiares com estas linguagens podem tornar-se rapidamente produtivos em *C#*.

O *C#* está disponível como parte do *Visual Studio .Net*. Proporciona acesso ao *Framework .Net*, que inclui uma máquina de execução comum e uma livraria de classes rica. O *Framework .Net* define uma *common language specification (CLS)*, um tipo de “língua franca” que assegura uma interoperabilidade continua entre as linguagens complacentes *CLS* e as livrarias de classes.

5.7.1 Hello World

O exemplo eterno do *Hello world* pode ser escrito em *C#* da seguinte maneira:

```
Using System;
Class Hello
{
    static void Main()
    {
        Console.WriteLine("Hello, world");
    }
}
```

Figura 5.28 – Código C# para o *Hello World*

O código fonte para o programa *C#* é tipicamente guardado em um ou mais ficheiros de texto com a extensão de *.cs*, tal como *hello.cs*.

Uma examinação próxima deste programa é bastante elucidativa.

A directiva **using System;** referencia um *namespace* chamado **System** que é providenciado pela livraria da classe *Framework .Net*. Este *namespace* contem uma classe **Console** que é referida no método **Main**. Os *namespaces* proporcionam uma maneira hierárquica de organizar os elementos de uma livraria de classes. Uma directiva “utilizada” permite uma utilização inqualificável dos tipos que são membros do *namespace*. O programa *Hello world* utiliza o bem mais pequeno **Console.WriteLine** em vez de **System.Console.WriteLine**.

O método **Main** é um membro da classe **Hello**. Tem o modificador **static**, e por isso é um método da classe **Hello** em vez de uma instância desta classe.

O ponto principal de entrada para um programa, é sempre um método estático chamando **Main**.

O *output* do *Hello world* é produzido através da utilização de uma livraria de classe. A linguagem não proporciona por ela própria uma livraria de classe. Em vez disso, utiliza uma livraria de classe comum que também é utilizada por linguagens como o *Visual Basic* e o *Visual C++*.

Para os programadores de *C* e *C++*, é interessante notar algumas coisas que não aparecem no programa *Hello world*.

O programa não utiliza um método global para o **Main**. Métodos e variáveis não são suportados ao nível global; tais elementos estão sempre contidos em declarações de tipos.

O programa não utiliza os operadores **::** ou **->**. O **::** deixou de ser um operador, e o operador **->** só é utilizado numa pequena fração dos programas. O separador **.** é utilizado em nomes compostos tais como **Console.WriteLine**.

O programa não utiliza o **#include** para importar texto do programa. As dependências entre programas são tratadas simbolicamente, em vez de textualmente. Este sistema elimina barreiras entre programas escritos em linguagens diferentes. Por exemplo, a classe **Console** podia ser escrita noutra linguagem.

5.7.2 Tipos

O C# suporta duas espécies de tipos: os tipos *value* e os tipos *reference*. Os tipos *value* incluem tipos simples (por exemplo: *char*, *int* e *float*), tipos *enum* e tipos *struct*. Os tipos *reference* incluem tipos de classes, tipos de interface, tipos delegados e tipos de *array*.

Os tipos *value* diferem dos tipos *reference* uma vez que as variáveis do tipo *value* contêm directamente os seus dados, e as variáveis do tipo *reference* guardam referências a objectos. Com o tipo *reference*, é possível duas variáveis referenciarem o mesmo objecto, e por isso é possível as operações numa variável afectarem o objecto referenciado pela outra variável. Com os tipos *value*, cada variável tem a sua cópia dos dados, e por isso não é possível as operações de uma afectar a outra.

5.7.2.1 Tipos predefinidos

O C# proporciona um conjunto de tipos predefinidos, sendo a maior parte familiar aos programadores de C e C++. Os tipos incluem os tipos *object*, *string*, *int*, *long*, *byte* e *temporal*.

5.7.2.2 Tipos de *arrays*

Os *arrays* podem ser unidimensionais ou multi-dimensionais.

5.7.2.3 Tipo *system unification*

O C# proporciona um “tipo de sistema unificado”. Todos os tipos – incluindo os tipos *value* – derivam do tipo *object*. É possível chamar métodos do *object* com qualquer valor, mesmo valores de tipos “primitivos” tais como o *int*.

5.7.3 Classes

As declarações de classes definem novos tipos de referências. Uma classe pode herdar de outra classe e pode implementar zero ou mais interfaces.

Os membros das classes podem incluir constantes, campos, métodos, propriedades, indexadores, eventos, operadores, construtores e destrutores. Cada membro tem uma acessibilidade associada, que controla as regiões de texto do programa que conseguem aceder ao membro. Existem cinco formas possíveis de acessibilidade:

Forma	Significado
<i>public</i>	Acesso não limitado.
<i>protected</i>	Acesso limitado a este programa ou tipos derivados da classe que o contém.
<i>internal</i>	Acesso limitado a este programa.
<i>protected internal</i>	Acesso limitado a este programa ou tipos derivados da classe que o contém.
<i>private</i>	Acesso limitado ao tipo que o contém.

Tabela 5.2

5.7.3.1 Campos

Um campo é um membro que representa uma variável associada a um objecto ou classe.

5.7.3.2 Métodos

Um método é um membro que implementa uma computação ou acção que pode ser realizada por um objecto ou classe. Os métodos têm uma lista de parâmetros formais (que podem estar vazios), um valor de retorno (ou **void**) e são estáticos ou não estáticos. Métodos estáticos são acedidos através da classe. Métodos não estáticos, que também são conhecidos por métodos de instância, são acedidos através de instâncias da classe.

5.7.3.3 Propriedades

Uma propriedade é um membro que proporciona acesso a um atributo de um objecto ou classe. Alguns exemplos de propriedades incluem, o comprimento de uma *string*, o tamanho de uma fonte, o *caption* (título) de uma janela, o nome de um cliente e por ai fora. As propriedades são uma extensão natural aos campos. Ambos são membros nomeados com tipos associados, e a sintaxe para aceder a campos e propriedades é a mesma. Contudo, ao contrário dos campos. As propriedades não indicam localização de armazenamento.

As propriedades são definidas com declarações de propriedades. A primeira parte de uma declaração é bastante parecida à declaração de um campo. A segunda parte inclui um *get accessor* e/ou um *set accessor*. No exemplo abaixo, a classe *Button* define uma propriedade *Caption*.

```
public class Button
{
    private string caption;
    public string Caption {
        get {
            return caption;
        }
        set {
            caption = value;
            Repaint();
        }
    }
}
```

Figura 5.29 – A classe *Button* define uma propriedade *Caption*

As propriedades que podem ser ambas lidas e escritas, tal como a propriedade *Caption*, incluem ambos o *get* e *set accessor*. O *get accessor* é chamado quando o

valor da propriedade é lido; o *set accessor* é chamado quando o valor da propriedade é escrito. Num *set accessor*, o novo valor para a propriedade é dado num parâmetro implícito chamado *value*.

5.7.3.4 Eventos

Um evento é um membro que permite a um objecto ou classe proporcionar notificações. Uma classe define um evento, proporcionando uma declaração do evento, que é muito semelhante à declaração de um campo mas com a palavra chave *event* adicionada.

5.7.3.5 Indexadores

Um indexador é um membro que permite um objecto ser indexado da mesma maneira que um *array*. Onde as propriedades permitem acessos do tipo campo, os indexadores permitem acessos do tipo *array*.

5.7.3.6 Destruidores

Um destrutor é um membro que implementa as acções necessárias para destruir uma instância de uma classe. Os destrutores não podem ter parâmetros, não podem ter modificadores de acessibilidade, e não podem ser explicitamente chamados. O destrutor para uma instância é chamado automaticamente durante a *garbage collection*.

5.7.3.7 Herança

As classes suportam herança simples, e o tipo *object* é a classe base derradeira para todas as classes.

5.7.3.8 Structs

A lista de semelhanças entre classes e *structs* é longa – *structs* podem implementar interfaces e podem ter os mesmos tipos de membros que as classes têm. As *structs* diferem das classes de algumas maneiras importantes, contudo as *structs* são tipos *value* em vez de tipos *reference*, e a herança não é suportada para as *structs*.

5.7.3.9 Interfaces

Uma interface define um contrato. Uma classe ou *struct* que implementa uma interface tem de aderir ao seu contrato. As interfaces podem conter como membros métodos, propriedades, indexadores e eventos.

5.7.3.10 Delegados

Os delegados permitem cenários que o C++ e outras linguagens tem endereçado com apontadores de funções. Ao contrário dos apontadores de funções, os delegados são orientados ao objecto e seguros.

Os delegados são tipos *reference* que derivam de uma classe base comum: **SystemDelegate**. Uma instância delegada, encapsula um método – uma entidade “chamável”.

Uma propriedade interessante e poderosa de um delegado, é que ele não sabe nem quer saber, sobre o tipo de objecto que referencia. Qualquer objecto serve; o que

interessa é que a assinatura do objecto seja igual á do delegado. Isto faz com que os delegados sejam perfeitos para uma invocação “anónima”.

5.7.3.11 Enums

Uma declaração de um tipo *enum*, define um tipo nome para um grupo relacionado de constantes simbólicas. Os *enums* são utilizados para cenários de “escolha múltipla”, nos quais uma decisão é tomada em *runtime* a partir de um número fixo de opções, que são conhecidas na altura da compilação.

A utilização de *enums* é superior á utilização de constantes *integer*, porque a utilização de *enums* torna o código mais legível e auto-documentável.

5.7.3.12 Namespaces e assemblies

Os programas apresentados até agora têm se mantido “sozinhos” excepto pela dependência de algumas classes, providenciadas pelo sistema, tais como a classe **System.Console**. É muito mais comum para um programa do mundo real, consistir de diversas peças diferentes. Por exemplo, uma aplicação corporativa, pode depender de diversos componentes diferentes, incluindo alguns que foram desenvolvidos internamente e alguns comprados a vendedores de *software* independentes.

Os *namespaces* e *assemblies* proporcionam este sistema baseado em componentes. Os *namespaces* providenciam um sistema lógico organizacional. Eles são utilizados como sistema de organização “interno” para um programa e como sistema de organização “externo” – é uma maneira de apresentar elementos de programas que estão expostos a outros programas.

As *assemblies* são utilizadas para empacotamento e distribuição física. Uma *assembly* actua como um contentor para os tipos. Uma *assembly* pode conter tipos, o código executável utilizado para implementar estes tipos e referencias para outras *assemblies*.

5.7.4 Atributos

O C# é uma linguagem procedural, mas como todas as linguagens procedurais tem alguns elementos declarativos. Através do seu suporte para atributos, o C# generaliza esta capacidade, para que os programadores possam inventar novos tipos de informações declarativas, especificar esta informação declarativa para diversas entidades do programa, e resgatar essa informação declarativa em *runtime*.

5.8 Forms Windows

A *forms Windows* é um novo pacote de *forms* que permite aos programadores construir aplicações baseadas no *Windows*, que tiram partido das características da interface de utilizador rica, disponível no sistema operativo *Microsoft Windows*. As *forms Windows* são parte do novo *Framework .Net* e tiram vantagem de muitas novas tecnologias incluindo um *framework* de aplicação comum, um ambiente de execução manejável, segurança integrada, e princípios de desenho orientado ao objecto. As *forms Windows* também oferecem suporte completo para uma conexão rápida e fácil a serviços web, e para construir aplicações ricas baseadas no modelo de dados *ADO .Net*. Com o novo ambiente de desenvolvimento distribuído do *Visual Studio*, os

programadores irão poder criar aplicações *forms Windows* utilizando qualquer uma das linguagens que suportam o *Framework .Net*, incluindo o *Visual Basic* e o *C#*.

5.8.1 Criar uma aplicação *forms Windows*

Os programadores irão criar aplicações *forms Windows* de uma maneira muito igual as *forms* baseadas no *Visual Basic* (embora com as *forms Windows*, tenham o mesmo nível de produtividade em todas as linguagens do *Visual Studio*, não só no *Visual Basic*). Na imagem que se segue, pode-se ver que a área de desenho é utilizada para expor visualmente a *form* com os *controls*. Para editar o código fonte, os programadores só têm que fazer um duplo *click* sobre o *control* e o editor de código fonte aparece, permitindo um acesso rápido ao modelo de eventos completo para o *control* e *form*.

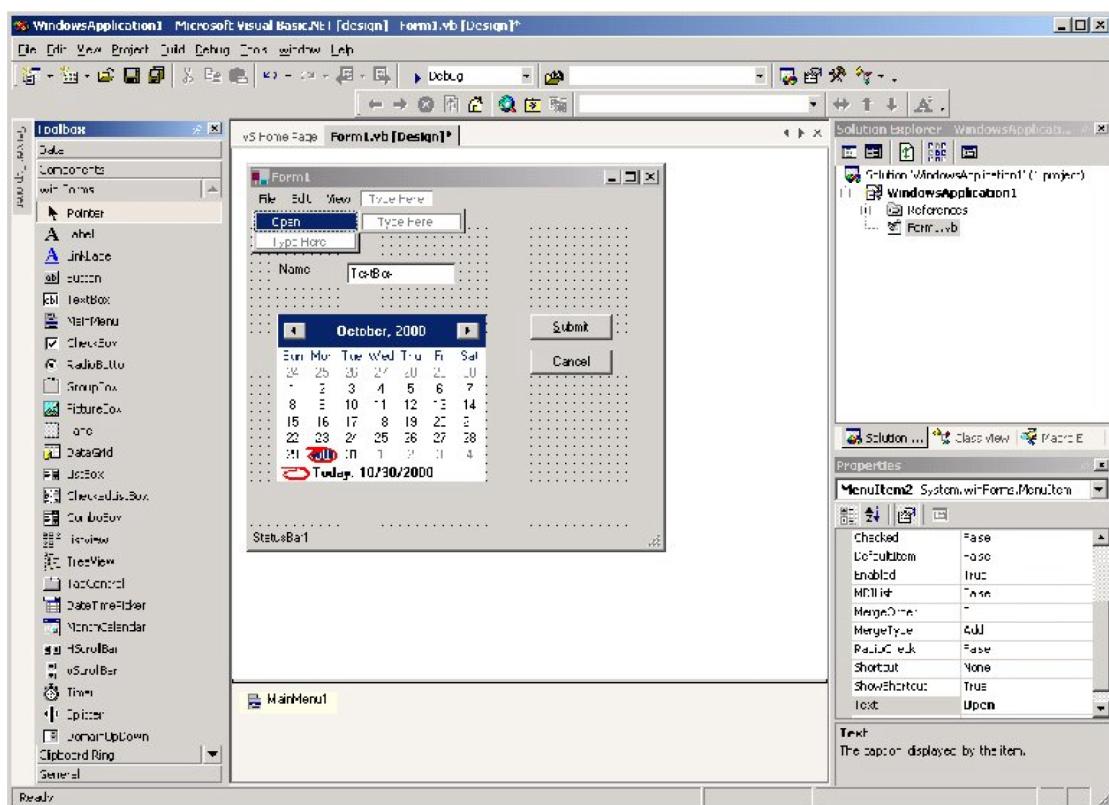


Figura 5.30 – Construir *forms Windows* utilizando o *Menu Designer*

5.8.2 Herança visual

A herança visual é uma das principais novas características disponíveis no *forms Windows*, que irá melhorar a produtividade do programador e facilitar a reutilização do código. Por exemplo, uma organização pode definir uma *form* base standard que contem itens, tais como o logotipo corporativo e talvez uma *toolbar* comum. Este *form* pode ser utilizado por programadores através de herança e pode ser estendida, para ir ao encontro dos requerimentos de aplicações específicas, enquanto promove uma interface de utilizador comum pela organização. O criador da *form* base pode especificar que elementos podem ser estendidos e os que têm de ficar como estão, certificando-se de que a *form* é reutilizada apropriadamente.

5.8.3 Desenho de *form* preciso

Os programadores irão ter um nível de controlo e produtividade sem precedentes, quando desenharem o *look* e *feel* da sua aplicação com as *forms Windows*. Características como o *Menu Designer*, *Control Anchoring*, *Control Docking* e muitos outros *controls* novos permitem um nível muito alto de poder e precisão para os programadores construírem interfaces de utilizadores baseadas em *Windows*.

Com o *Menu Designer*, os programadores podem rapidamente e facilmente adicionar menus a uma *form*, modificá-los e visualizar como irão ficar, sem ter de executar a aplicação. Os *controls* na *form* são mais eficazes com o *Control Anchoring*, permitindo a uma *form* fazer o *resize* automático dos *controls*, conforme um utilizador faz *resize* ao *form*. Com o *Control Docking*, os *controls* podem ser “presos” a qualquer lado da *form*, proporcionando uma maior flexibilidade no *layout*.

Controls ActiveX existentes, podem também ser utilizados e “corridos” em qualquer *form*, preservando assim investimentos em tecnologias existentes.

Os *Controls* novos – incluindo o *Link Label* – proporcionam funcionalidades adicionais comuns para os programadores. O *Link Label* proporciona *liking* do tipo *HTML* para um *URL* específico. O texto exibido utilizando este *control* irá aparecer sublinhado e o cursor irá mudar para uma mão conforme o rato se mover por cima dele, disparando assim um evento de acção quando é “clickado”.

5.8.4 Um custo total de posse, mais baixo

As *forms Windows* proporciona mais do que uma óptima maneira de construir aplicações baseadas no *Windows*. Os programadores também beneficiam de capacidades de distribuição mais fáceis e de um modelo de segurança de aplicação integrado. As *forms Windows* tira partido das características de *versioning* e distribuição da plataforma *Microsoft .Net*, para oferecer, com o correr do tempo, custos de distribuição mais reduzidos e uma robustez mais elevada. Isto reduz significativamente os custos de manutenção para as aplicações escritas em *forms Windows*.

Com uma aplicação *forms windows*, não há necessidade de distribuir uma aplicação para o *desktop* do utilizador final. Em vez disso, o utilizador pode invocar a aplicação escrevendo simplesmente o *URL* num browser. A aplicação irá ser “carregada” para a máquina cliente, executada num ambiente de execução segura, e remove-se sozinha quando a execução termina.

Para organizações que querem distribuir fisicamente uma aplicação para o *desktop*, não existe a necessidade de percorrer um processo de instalação que consome imensos recursos. Basta copiar a aplicação para o *PC* e já está tudo pronto. As *forms Windows* proporcionam a flexibilidade para tornar as aplicações disponíveis aos utilizadores finais, da maneira que é mais apropriada ao negócio.

5.9 Quais são os requisitos do sistema para o *Visual Studio .Net Beta 1?* (Hardware e software)

A Microsoft diz, que os requisitos do *Beta 1* são significativamente mais altos, e a sua execução significativamente mais baixa do que a versão final do *Visual Studio .Net*. A versão *Beta 1* do *Visual Studio .Net*, foi testada no *Windows 2000 (Server e Professional)*, no *Windows NT 4 (Server e Workstation)*, no *Windows Millennium Edition* e no *Windows 98*. A configuração de hardware recomendada, é a seguinte:

Requisitos mínimos <i>Beta 1</i> *	Requisitos recomendados <i>Beta 1</i> *
Processador: PII, 450 MHz	Processador, PIII, 733 Mhz
RAM: 128 MB	RAM: 256 MB
Placa de vídeo capaz de 800x600, 256 cores	Placa de vídeo configurada para 1024x768, Hi-Color
1 GB disponível em disco	1GB disponível em disco
Leitor de CDs	Leitor de CDs

* Estes requisitos são só para *Beta 1*.

Tabela 5.3

As novas características do *Visual Studio .Net* fazem dele um ambiente de desenvolvimento completo para construir no *Framework .Net* da Microsoft, a plataforma de desenvolvimento de aplicações web da Microsoft da próxima geração. Ele providencia tecnologias chave para simplificar a criação, distribuição, e a evolução constante de aplicações web e serviços web XML seguros, escalonáveis e disponíveis enquanto aproveita toda a perícia do programador.

6 Segurança

O *Framework .Net* inclui uma grande variedade de características de segurança que estão em proporção à amplitude da própria plataforma. Quatro das áreas mais importantes são:

- segurança baseado em função;
- segurança de aplicações web;
- segurança baseada em factos e
- criptografia.

6.1 Segurança baseada em funções

O *Framework .Net* introduz um modelo unificado para manusear a identidade e função do utilizador (ou agente autonomizado), para uma autorização. O modelo é baseado no conceito de que um utilizador é um *principal*, a favor de quem, código é executado.

A autenticação, é o processo de examinar credenciais (por exemplo, nome/*password*) e de estabelecer a identidade do *principal*. Para além da identidade, um *principal* pode pertencer a zero ou mais funções, representando autorizações. O código da aplicação pode então conhecer a identidade do *principal* corrente, ou questioná-lo para uma qualquer função na qual seja necessária executar alguma operação privilegiada.

Para organizações, a identidade do utilizador de *logon* para o *Windows* é uma forma importante de identidade para a segurança, por isso o *WindowsPrincipal* é disponibilizado para se encarregar da autenticação. O *username* do *Windows* torna-se a identidade *principal*, e os grupos aos quais o utilizador pertence, são os nomes e funções designados.

6.2 Segurança de aplicações web

A *ASP .Net* foi construído com a segurança em mente. A *ASP .Net* dá poder ao *Internet Information Server (IIS)* para proporcionar um suporte forte, para esquemas de autenticação *HTTP* comuns incluindo suporte para *Basic*, *Digest*, *NTLM*, *Kerberos* e certificados cliente *SSL/TLS*. A *ASP .Net* também suporta a autenticação *Microsoft Passport*, e proporciona uma implementação conveniente de autenticação baseada em *Forms (Cookie)*. Independentemente do esquema de autenticação aplicado, os programadores têm um modelo consistente de programação e autenticação.

A *ASP .Net* suporta métodos tradicionais de executar controle de acesso e também proporciona autorização *URL*, o que permite aos administradores proporcionar configuração *XML*, que dá ou nega acesso a *URLs* baseados no utilizador ou função corrente. Os programadores podem facilmente codificar verificações de autorização explícitas, nas suas aplicações ou podem tirar partido do suporte da *common language runtime*, para que a segurança declarativa, inclua acesso ao controle a métodos, baseados no utilizador ou função de chamada.

A *ASP .Net* tem uma arquitectura de segurança muito extensa, que permite aos programadores escrever autenticações comuns ou *providers* de autorizações. Estes *providers* podem tratar de eventos de autenticação e autorização no ficheiro *global.asax* ao nível da aplicação, ou os programadores podem escrever um módulo, que pode ser reutilizado por outras aplicações.

6.3 Segurança baseada em factos

Para além da confiança nos utilizadores, a confiança no código com as subsequentes restrições enforcadas, também é crítica, para se obter uma boa segurança no espaço da aplicação *.Net*. O código manejável a executar no *Framework .Net*, pode ser restringido a só poder utilizar interfaces bem definidas, o que permite a segurança ser efectivamente enforcada no código. Isto permite a grandes aplicações compostas de muitos componentes, a serem distribuídas com segurança e com variados graus de segurança enforcados aos variados componentes. Isto permite um número de possibilidades que não estavam disponíveis ás aplicações anteriormente:

- código móvel pode ser *downloaded* de origens inseguras e executadas em segurança com restrições;
- as aplicações de servidor podem ser “estendidas” com código escrito pelo utilizador, que corre forçado a não interferir com a operação geral do servidor;
- aplicações programáveis podem executar com segurança macro *script* associadas a documentos de utilizador.

Antes de que qualquer código manuseado, seja executado, a política do sistema de segurança determina que permissões irá atribuir, baseadas em factos, no conjunto do código e nos pedidos do próprio código. Os factos podem ser qualquer coisa conhecida sobre o código; uma qualquer assinatura digital, o *URL*, *site* ou zona da qual o código é proveniente, e por ai fora. As políticas de segurança configuradas pelo administrador ou utilizador, especificam regras de utilização de factos, para determinar permissões a atribuir ao código. Depois de assegurar que as permissões mínimas, que o código pode requisitar, podem ser dadas, e excluir permissões que o código não quer, as permissões são atribuídas e o código é executado , limitado pelo que as permissões o permitem fazer.

6.4 Criptografia

O *Framework .Net* inclui funções criptográficas para incriptação, assinaturas digitais, *hashing*, e geração de números *random*. Os algoritmos suportados incluem: incriptação assimétrica – *RSA* e *DSA*; incriptação simétrica – *DES*, *TripleDES*, *RC2*; *hashes* – *MD5*, *SHA1*. A implementação inclui um modelo baseado em *stream* (fluxo) por isso um *stream* de dados de um ficheiro, pode por exemplo, ser dirigido para um objecto de incriptação e o *stream* resultante enviado para a rede.

Também está em estudo (na *Microsoft*) a especificação proposta *XML Digital Signature (XMLDSIG)*, que está de momento em desenvolvimento pelo *IETF* e o *W3C*. A *XMLDSIG*, proporciona uma maneira fácil de os programadores de aplicação, assinarem documentos e fragmentos *XML*. O *XML* assinado (e no futuro, incriptado)

será cada vez mais importante, como uma maneira de enviar, pela *web*, mensagens assíncronas.

7 Conclusão

Se o .Net for tudo aquilo que promete ser então, é possível que se tenha achado a solução ideal para a total integração da Internet no dia-a-dia.

Contudo, nem tudo é um mar de rosas e existe a necessidade, por muito pouca que seja, de uma integração, quer do conceito .Net em si, quer das próprias pessoas que se irão deparar com a nova plataforma (programadores, clientes, etc.). Irá sempre aparecer alguém que não vê qual é a necessidade para a nova plataforma. Só precisamos de nos lembrar, que quando apareceu o *Windows*, também ninguém precisava dele!

Quer a *Microsoft*, consiga ou não, entregar o .Net a tempo, as fundações e conceitos em que o .Net é baseado, são sólidos e estão para ficar. Quer seja a plataforma .Net ou outra qualquer plataforma, que possa eventualmente aparecer, o conceito é para ficar.